Leveraging the Benefits of Physical Computing in a Beginner-Focused Programming Environment

Sébastien Dunne Fulmer

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Engineering (Computer Engineering)

Supervisor: Assistant Professor Glenn Strong

May 2024

Leveraging the Benefits of Physical Computing in a Beginner-Focused Programming Environment

Sébastien Dunne Fulmer, Master of Engineering

University of Dublin, Trinity College, 2024

Supervisor: Assistant Professor Glenn Strong

The availability of affordable and varied physical computing devices designed for hobbyists and students has led to an increased interest in studying how these devices may be used to enhance computer science education, with a particular focus on engaging female students. This dissertation explores how a physical computing device may be incorporated into an existing educational programming environment, leveraging many of the reported benefits of such devices while addressing some of the identified deficits and expanding the activities and projects feasible to deliver in a classroom environment with inexperienced users. Support was implemented for an affordable device with various features used in many schools, the BBC micro:bit, into the Pytch programming environment, which targets inexperienced users transitioning to text-based programming. The approach was evaluated through two interventions conducted with Irish secondary school students, suggesting benefits to this approach in line with previous studies on physical devices. The dissertation also explores potential options for further development and evaluation of the approach taken in order to improve and better understand its effects.

Acknowledgments

I would like to thank my supervisor, Glenn Strong, as well as the other members of the Pytch team, Sara Fiori, Ben North and Duncan Wallace, for all of the support and guidance they provided over the course of the project. I am grateful for their advice, assistance and encouragement, which helped to ensure the completion of the project to a standard I was satisfied with.

I would also like to thank my fellow engineering students and close friends, Erin and Oscar, who helped me to survive my studies during my time in college and even made them fun at times, and the great friends I made in the Hist, Caoimhin, Daniela, Sinziana, Anna and Katy, who gave me a space away from my course to enjoy my college experience, and to learn and develop myself in another way. The extent to which I got involved in college life would not have been possible without the advice and support of Luke, Kayleigh and Professor David McConnell, to whom I also owe a large amount of gratitude.

Finally, I would like to thank my mum and dad for their love and support throughout my entire time in college and before it, which enabled me to develop and thrive at Trinity.

Sébastien Dunne Fulmer

University of Dublin, Trinity College
May 2024

Contents

List of Tables

List of Figures

Acronyms

API Application Programming Interface.

BBC British Broadcasting Corporation.

CPSES Computer Programming Self-Efficacy Scale.

CT Computational Thinking.

DAL Device Abstraction Layer.

GPIO General Purpose Input/Output.

HID Human Interface Device.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

IMMS Instructional Materials Motivation Survey.

JES Jython Environment for Students [7].

JSON JavaScript Object Notation.

JSX JavaScript XML.

LED Light-Emitting Diode.

PDF Portable Document Format.

RGB Red Green Blue, often in the context of addressable multicoloured LEDs.

SCSS School of Computer Science and Statistics, Trinity College Dublin.

SMQ-II Science Motivation Questionnaire II.

SPA Single Page Application.

SPI Serial Peripheral Interface.

SWD Serial Wire Debug.

UART Universal Asynchronous Receiver-Transmitter.

USB Universal Serial Bus.

VM Virtual Machine.

XML Extensible Markup Language.

Chapter 1

Introduction

Just like the field itself, Computer Science Education is a rapidly evolving area of research and practice. Some countries, such as the United States and the United Kingdom, have established technology and computer science curricula at the pre-university level, however others, such as Ireland, have only recently begun developing such programmes. In many instances, individuals may only be introduced to Computer Science at a late educational stage, when they may already be familiar with many mathematical and physical concepts. Appropriate computer science education needs to be capable of maintaining the interest of these individuals while still catering to their level of experience, and effectively achieving learning outcomes.

The project aims to help tackle both elements through employing the use of physical computing devices. Physical devices can open up new possibilities in the types of tasks that can be accomplished through programming, especially when it comes to interacting with the physical world around a student, and the student themselves. These new possibilities are often able to catch a variety of individuals' interest, and ground the concepts they are learning in tangible real world applications. By incorporating physical computing into an approachable programming environment, such as Pytch, the motivational and educational benefits of using such devices can be more easily accessed both in formal educational settings and directly by interested students.

1.1 Pytch

The project will build upon the pre-existing Pytch environment to introduce physical computing capabilities. Pytch is a programming environment designed to support students making the transition from block-based languages, such as Scratch, to text-based languages such as Python, being developed in the School of Computer Science and Statistics of Trinity College Dublin in collaboration with the Technological University of Dublin

[27]. Pytch enables students to continue using the simple programming paradigm encountered in an environment like Scratch with actor-based programming and concurrency when transitioning to conventional programming languages like Python. This enables an inexperienced user to develop more engaging programs without the complexity of managing systems like graphics, audio input and concurrency, and thus facilitates more engaging lessons while still having the user write correct and idiomatic code which could be used in other contexts outside of the environment. This approach makes the transition more gradual by allowing the student to focus on the differences and additional capabilities of modern text-based programming languages without worrying about these additional systems, before then transitioning towards a more traditional Python environment once they have achieved a sufficient degree of proficiency [27]. Pytch is primarily targeted towards educational use in schools, and has been trialled in multiple schools in the Republic of Ireland during its development, which are representative of the project's target audience as a country with a newly established computer science curriculum.

1.2 Motivation

There are deficiencies both within Ireland and more broadly in computer science education. Computer science curricula often do not put sufficient emphasis on the transitional period from introductory tools and environments which are designed to make computer science concepts broadly accessible, and the traditional tools and environments which are commonly used in industry and later education. This poses a major obstacle to students who are interested in computer science but may lack the available support or natural inclination to overcome this gap by themselves. This effect can be more pronounced for individuals who are historically excluded from these fields and so are less likely to have access to this support or be introduced to computer science at an early age, such as women and ethnic minorities [17]. Additionally, this stage of computer science education can be relatively unengaging for a student, as it will often focus on basic data structures and algorithms, rather than the more tangible projects with rapid gratification that are typical of introductory courses [15, 27]. If the work is particularly difficult and the student is not receiving gratification from their work, then it is far more likely that they may cease studying computer science [17]. Environments which target this transitional period, such as Pytch, can make a substantial impact in addressing these problems. By expanding such environments into the world of physical computing, they can become even more effective by providing a bridge between the environment and the physical world, leveraging the associated expanded opportunities and motivational benefits [8, 24].

1.3 Goals and Objectives

The overall goal of the project is to implement support for utilising a readily available computing device with a broad set of features into a programming environment oriented towards inexperienced users and educational use, Pytch, and to evaluate the impact of this addition on student motivation and learning outcomes for secondary school students studying computer science.

The goal has been broken down into a series of more specific objectives:

- Research available physical computing devices and evaluate them with regards to their suitability for use by inexperienced users and in a school environment in order to select a device or devices to support.
- Implement functionality into a programming environment in order to connect to, setup and communicate with the chosen device through the user's browser.
- Develop the necessary software for the chosen device in order to provide a communication protocol enabling interaction with the device from Pytch via an appropriate communication method.
- Design and implement an interface that is easy to use for an inexperienced user without extensive instruction for pairing new devices and managing connected devices.
- Implement a library within Pytch which exposes the functionality from the chosen device through an understandable and idiomatic interface that is both compatible and consistent with other pre-existing libraries in Pytch.
- Develop one or more sample activities which make use of the chosen device's functionality for use by users or instructors directly or as a reference for developing further activities.
- Evaluate the efficacy of the implementation regarding student motivation and learning outcomes, by delivering a sample activity to a group of inexperienced students as part of a study and analysing the collected data.
- Evaluate the end outcome of the project against its goal and objectives, including both the technical implementation and any supplemental activities or materials developed, with reference to prior research in the area and alternative solutions.

1.4 Approach

The project will start with a preliminary research and design phase. A literature review will be conducted in order to identify relevant research regarding approaches to computer science education, existing educational environments and previous studies incorporating physical devices. This research will inform the design of the project in order to achieve the best results, and will be expanded upon throughout the project. The potential options for physical computing devices to support will also be researched and evaluated. Once a sufficient level of research has been conducted, a device will be chosen to support, and a compatible connectivity method will be identified to implement support for the device through.

The technology necessary to incorporate the selected device into the chosen environment, Pytch, will then be implemented, and a sample activity leveraging the device will be designed. Activity design will involve a general exploration of potential activities, an expansion of promising ideas, and the selection and development of one or two sample activities. Given that evaluation will be conducted in an Irish context, it is important that the areas covered by the Irish Leaving Certificate Computer Science curriculum are considered when selecting and designing an activity.

Finally, an intervention using the implementation will be designed and conducted, with the results and the implementation of the project being evaluated. The outcome of the project will be compared against its goals, potential refinements to the implementation will be suggested, and the efficacy of the implementation will be assessed. The details of the project, including the analysis and results, will be written up in the dissertation.

1.5 Gender Dimension

It is important when examining computer science education to be cognisant of the historical and continued underrepresentation of women in both computer science professions and academic courses. The issues with academic and professional culture and methods, as well as gender norms and expectations imposed from an early age, and a general lack of support in the pursuit of computer science for these individuals have all been well documented [17, 21, 28]. Constructivist, practically relevant and creative approaches have all shown evidence of engaging and retaining female students in introductory and extra-curricular computer science programmes [3, 21, 26]. As has been acknowledged in the motivation and will continue to be developed further, the Pytch project occupies an important space as a transitional environment in a prospective student's computer science education, from purely educational environments like Scratch towards textual programming languages

used in the professional world. Due to its orientation towards inexperienced users, Pytch may also find use as an introductory environment for users with no prior experience with computer programming. As a multimedia focused environment leveraging some of the same benefits seen with Scratch relating to rapid prototyping and relevant use, Pytch already addresses some of these concerns to a notable degree. However, extending this effect further, into the subfield of physical devices and embedded programming, as well as broadening the possibilities available within Pytch, making it more widely applicable and able to be engaged with for longer, is likely to amplify this effect for affected individuals already engaging with Pytch and help reach a broader group of those individuals. Confirming, quantifying and amplifying these effects is a substantial task that would require study in its own right, however prior research and studies indicate that this type of approach could have a positive impact, and examining this in more detail would be a potential area for further study after the completion of this initial implementation.

1.6 Outcome & Contributions

The project was successfully implemented, achieving its goal and all of its objectives, as well as meeting the requirements outlined in section 3.1.

Two interventions were conducted, one with a group of students without prior experience of the Pytch environment in the form of a workshop as part of a larger event, and one as a study conducted with a group of students with prior experience of the Pytch environment. The evaluation of these interventions found that the approach and implementation were effective at motivating the students involved, and identified multiple areas for further development or study. The results of the interventions also further validate previous findings that physical devices and experimental approaches at learning are especially effective at engaging female students and students with limited prior experience with computer science. The technical components of the project are planned to be merged back into the broader Pytch project later this year, and the sample activity and lesson developed as part of the project is planned to be made available through the Pytch website with some modifications.

From a research perspective, the project has contributed to the ongoing research into physical computing and educational programming environments in computer science education that has been ongoing for multiple decades. Much of this research has been conducted in areas with more established technology curricula, while this project is conducted in a context with many individuals only experiencing formal technological education in their mid-to-late teenage years. Additionally, multiple prior studies have found difficulties in setting up physical devices to be a major roadblock, which is one issue the project

seeks to address. The outputs of the project will also assist the broader Pytch project with its ongoing research by expanding the opportunities for use of the environment, and enabling the previously researched benefits of physical devices for the Pytch environment. The evaluation and conclusions reached in the project can help to inform the future research and development of the Pytch environment, as well as the study of educational environments and device-based education in computer science.

1.7 Structure of the Dissertation

This initial chapter has introduced the necessary material to understand the purpose and content of the dissertation.

Chapter 2 will explore the pre-existing literature relating to both the use of physical interfaces and the use of beginner-friendly or transitional programming environments in computer science education, examine the relevant pre-existing aspects of the Pytch programming environment, the available physical computing devices, and similar pre-existing projects.

Chapter 3 will explore the design process for the project, including the selection of a suitable physical computing device, the design of the different technological components, the design of the sample activities, and the design of the evaluation process. Chapter 4 will then explore the implementation of the project under similar headings, along with the refinement process that was followed between feedback sessions and trials.

Chapter 5 will explore how the project was evaluated, including a comparison of the end outcome to the previously stated goal and objectives and the results of the trials that were conducted. Chapter 6 will then draw conclusions from these evaluations, reflecting on the implementation of the project, and explore how the implementation and evaluation of the project could be improved upon or expanded in the future.

Chapter 2

State of the Art

Through a review of the relevant literature, this chapter will explore relevant theories behind computer science education, the value of educational environments like Scratch and Pytch, how physical approaches can be leveraged to enhance computer science education, and how these approaches can help encourage diversity in the field. This chapter will also examine the relevant aspects of the technical architecture of Pytch necessary for understanding the design and implementation of the proposed solution. The available physical computing devices which could potentially be selected for use in the project will also be analysed. Finally, similar existing solutions will also be examined in order to inform the design discussed in chapter 3 and to demonstrate the uniqueness of the approach taken by the project.

2.1 Literature Review

There has been extensive research into different approaches to teaching computer science, with a large emphasis on the educational theory of constructivism, which appears to be especially applicable to computer science. Recent advancements in physical computing devices, like microcontrollers and single-board computers, and a general reduction in their cost have led to a renewed focus on researching the benefits of their usage, with a specific focus on how they interact with diversity in the field [9]. In this review, the theories relevant to the project will be discussed in the context of the Pytch environment, before examining some prior research and experiments in the same area, and exploring the specific benefits around encouraging diversity.

2.1.1 Computer Science Education

Modern approaches to computer science education are firmly rooted in the educational theory of constructivism, which "claims that students construct knowledge rather than merely receive and store knowledge transmitted by the teacher" [2]. This paper states that a major obstacle to success in the study of computer science is often the lack of an effective model of how a computer functions, however constructing one is particularly accessible through effective instruction. It posits that the frustration and difficulty of becoming acquainted with computer science arises from the fact that such a model often has to be constructed from the ground up. The related idea of 'bricolage' involves taking materials and solving problems through trial-and-error and experimentation, and is proposed to combat high attrition in introductory courses, and especially the underrepresentation of women and ethnic minorities [26]. The core idea of a 'bricolage approach' presented in this paper is that students are provided with an existing program which they then adapt to achieve a new behaviour through experimentation and exploration, building their computational model in a constructivist manner. The paper recognises that it may take longer to reach a similar level of familiarity or understanding through bricolage compared to a traditional approach, but that it achieves its aims of keeping less technically experienced students involved, especially female students, which is particularly valuable in introductory courses.

A related consideration is the relevance of material to students and prospective students. For individuals without a technical background, introductory computer science courses can be seen as overly technical, lacking relevance to the real world, boring and uncreative [7]. One approach to introducing computer science concepts, 'media computation', focuses on teaching core concepts through the manipulation of digital media, such as images, video and audio. To support this effort, the researchers constructed a multimedia Python environment written in Java, the 'Jython Environment for Students' (JES). Digital media was seen as a particularly relevant theme as most individuals interact with it constantly in their daily lives and manipulating this media is an obviously useful skill. Additionally, digital media is a viable method to introduce core concepts, such as data structures, loops and conditionals, which are obviously applicable to digital media files. Since these concepts are being learned through a real text-based programming language, Python, they are transferable to more advanced courses and other areas, allowing students to move beyond the specific environment. An important consideration which recurs throughout research about both bricolage and media computation is the idea that, in order to get satisfying and engaging results, a large amount of cumbersome programming is required which is not motivating to beginning students. JES helps to address this by providing visualisation tools and functions to simplify loading, working with and display-

ing or playing media. This approach was effective at reaching inexperienced students and underrepresented groups similar to a bricolage approach, by making computer science seem more approachable, creative and relevant [7, 21], and has been adopted or adapted in other tools and studies [5, 16].

As computer access, literacy and instruction becomes more common, many children engage with forms of programming of their own accord without necessarily conceiving of it as such [20]. Children find interest in experimenting with and modifying programs, whether that is intended modification at a basic level, such as a customisable video game, or at a more technical level using simple game engines or programming environments. An intrinsic motivation drives the children to engage in this behaviour and so it is not seen as tedious. Many of these environments which are distributed freely are designed with tinkering or experimentation in mind, including engaging example programs and encouraging exploration of and experimentation with these programs. Children typically learn these environments by trying things out, starting with the examples and only resorting to tutorials when they are stuck if they cannot ask someone for help. As a result, it tends to be the environments that provide useful examples which serve as springboards for interesting projects that tend to be the most successful with children. The ability for a child to set up and produce early results quickly are also major factors in an environment's success with children, similar to the ideas motivating bricolage and media computation as approaches to computer education. Children also tend to spur each other on and want to share their creations with others, often receiving a form of social reward as well as constructive suggestions for improvement or ideas about what else might be possible [20].

A common idea in environments targeted towards children are 'microworlds', "environments where people can explore and learn from what they receive back from the computer" [11]. Learners interacting with microworlds are "in the position simultaneously of user and designer", which is also often the case in professional computer science, and facilitates constructivist learning. Microworlds can be separated into the 'superstructure', the objects and tools provided by the microworld, and the 'platform', where new objects and tools can be created, and the behaviours of existing ones altered. The ability to descend past the superstructure to the platform is considered an important aspect of what makes microworlds an effective learning tool, allowing the learner to move beyond the constraints and create something new from the program provided. In particular, this paper examines games as microworlds, providing a game built in a visual programming environment and having young children manipulate the superstructure before ultimately altering it at the platform level. Of particular interest is how enjoyment of the game affects the motivation to alter it. In the case study, finding the provided game too boring provided a motivation to improve the game by modifying it, making it more compelling,

in part due to the fact that the children had made it their own. Their customisation of the game made it more satisfying to play, making the action of customisation itself satisfying. The ability for the children to take the program apart, examine how it functioned and observe how it functioned after making changes was essential to their modifications and was aided by the visual analogies provided by the platform the game was built on; an effective bricolage approach in action.

Related to microworlds is the 'Use-Modify-Create' methodology, where a student progresses from using and exploring an existing program, to making increasingly sophisticated modifications to that program, before moving on to developing their own projects with the skills and confidence developed [14]. The 'Modify' step has distinct parallels with the conception presented of a productive microworld, where learners first work at the superstructural level, altering the objects provided in simple ways, and later move to the platform level to make significant changes, furthering their understanding as they go. Proposed as an effective methodology for building 'computational thinking' (CT) skills in younger students, Use-Modify-Create has seen substantial use in various studies, often leveraging microworld-style environments such as Scratch. A structured approach to interrogating and reflecting on the provided code and how it functions has been shown to help build understanding about more abstract computer science concepts [22]. This further indicates that a bricolage approach such as Use-Modify-Create can be an effective way of constructing a viable computing model at the introductory level, alongside improving student confidence and motivation.

With technological advancements and decreasing cost across a range of different physical computing devices making them more accessible for hobbyists and educators, the use of physical devices in computer science education has been an area of substantial research [9]. Various studies have indicated that physical devices can help facilitate more relevant lessons and deeper motivation by engaging students in more practical tasks with visible results, as well as enabling a greater degree of creativity. Working on or through a physical device operates as another form of constructivist learning, either experimenting with how the device physically works or observing how a computer program responds to its manipulation, amplified by links between physical activity, the sensory system and cognition [18]. One method of categorising physical computing devices is based on their function and connectivity, from packaged electronics with no programming in category 1 to standalone devices which can be programmed and integrated into complex electronics projects in category 5 [9]. Recent research has centred around devices in category 4 of this division [3, 8, 24], which are devices programmed using computers but are subsequently used independently [9]. The motivational benefits have been consistent across these different studies, as well as the benefits for groups historically underrepresented in

computer science, such as women and ethnic minorities. A factor worth noting is that tangible interfaces can be more inviting to students but can struggle to gain adoption due to the greater financial and labour investment required [10]. This paper suggests that this obstacle can be overcome with a hybrid system, where both virtual and physical elements are used together, either at the same time or as different methods of interacting with a single system. This enables an individual to evaluate the system before making a major investment, which may make them more willing to make such an investment, as well as enabling more possibilities for the system overall and the idea of eased progression between different parts of the system as a result of increased familiarity.

2.1.2 Examining Scratch & Pytch

Scratch is a block-based programming environment developed at MIT's Media Lab with a focus on enabling interactive and media-rich projects [16]. Scratch also provides a social component for its users, enabling them to share their programs, receive feedback and learn from others, which can be a particularly powerful motivator for children as discussed previously. Scratch is an "always live" environment [16], meaning that there is not a compilation step in the conventional sense as code can be immediately run and its effects observed, and Scratch also allows for triggering individual pieces of code to instantly see their effect. In this way Scratch projects form a microworld, as their components can be interacted with directly on the stage and modified in the sprite editor, but their behaviours can be inspected or modified and new objects can be created. This is part of what makes Scratch suited to the Use-Modify-Create methodology, something Scratch enables naturally through its social components, enabling users to find projects they enjoy or which implement a certain behaviour and disassemble them to understand how they work. The motivational effect of instantly visible rewards and a fast feedback loop has been demonstrated to be effective even at the university level and is one of the reasons why Scratch has been effective and prolific as an introductory programming tool [15]. Scratch is able to introduce the core concepts of computing, like variables and control flow, in a more enticing environment, which then eases the transition on to other environments, similar to the motivation and approach of media computation. As a media-focused environment, there are a lot of parallels between the strengths of media computation and Scratch, but Scratch is able to go further into enabling interactivity, and thus games, which can be especially motivating and satisfying for younger users.

The main weaknesses of Scratch tend to be the limitations it places on more experienced individuals and the relatively sharp transition from a block-based multimedia environment to a text-based programming language without readily accessible multimedia

capabilities [15]. The Scratch language is intentionally simple, missing various data structures present in traditional programming, and the method of interacting with its objects is not obviously transferable to other contexts where these capabilities are not readily available as part of the environment. The Pytch project addresses both of these weaknesses by taking a Scratch-like environment with strong multimedia capabilities and actor-based concurrency, and enabling users to program in Python through a method more similar to traditional text-based programming [27]. Pytch positions itself as a bridge from the block-based programming present in environments like Scratch towards traditional text-based programming, but is also potentially viable as an introductory environment in its own right for the same reasons Scratch is. By providing a similar environment Pytch reduces the cognitive load of the transition to by allowing the learner to focus on learning more complex text-based programming, rather than the libraries and systems involved in rendering graphics or handling concurrency, similar to JES. Pytch's similarity to Scratch and low barrier to entry for new users is an important aspect of its efficacy and is enshrined in Pytch's five design principles: supporting the Scratch concurrency model, supporting the Scratch 'micro-world' model, encouraging the use of idiomatic Python code, minimising the need for complex set-up and administration, and supporting user autonomy via integrated learning materials. Additionally, the interface exposed by the Pytch environment uses similar terminology and phrasing to Scratch, which helps to make the transition easier for users familiar with Scratch.

2.1.3 Physical Approaches

One type of physical computing employed in education for an extended period of time is robotics. Robotics projects and kits of varying complexities and functionalities have been available and used for a number of decades as a method of teaching electronics and computer programming. Two studies conducted in Carnegie Mellon University explored designing a robotics system specifically for use in introductory computer science [12, 13]. These studies identified that previous attempts were held back by the cost of robotics kits and limited lab time restricting students' ability to work on the robots, coupled with observation-based debugging introducing delays in development. These studies sought to develop a system that allowed for every student to be loaned or own a robot while still being adequately capable for the course. In the initial trial, 'CSbots', an iRobot Create robotics platform with a Qwerk controller was used, which provided a rich set of features but were not cheap enough to be loaned out [13]. Only one trial of this initial system was conducted, but there was positive feedback on interest, twice as many students completed all the assignments compared to previous years despite their low contribution towards the

overall grade, and exam performance also improved compared to the previous 2 years.

The second trial involved the 'Finch' robot, which aimed to achieve a lower cost and be portable enough for students to take home [12]. One major change was utilising a tethered approach, where the robot platform only exposes the hardware and a computer connected via USB performs the actual processing, making the device easier to set up. This moves the robot from category 4 to category 3 device in the device categorisation discussed previously [9], essentially becoming a physical peripheral for a computer. This enabled the Finch to hit a more affordable estimated cost of 100 US dollars. The Finch was programmed in Java, and to keep the interface simple all functionality was exposed via a single class. The Finch was formally piloted in an introductory course and a follow-on course, as well as being loaned out to a variety of schools, after-school programmes and individuals, where it was generally positively received. The Finch was not observed to have a substantial impact on grades for the introductory course but did have a positive impact on the follow on course, and most students preferred it to programming solely on a computer. The tethering approach is of interest, as it made it easier to start working with the Finch and brought it closer to being a hybrid approach. While there may be drawbacks to this approach with a mobile robot, in the context of other physical devices it may be more suitable.

Another tethered solution was 'Phidgets', which aimed to develop a physical equivalent of user interface widgets, physical components with consistent interfaces which can be easily incorporated into applications [6]. As a result, Phidgets also can be considered a category 3 device in the discussed categorisation, operating as peripheral devices for a computer but with an electronics focus, incorporating sensors and digital inputs and outputs more directly [9]. In the Phidgets study, 16 undergraduate students with a computer science background but no experience working with physical devices were tasked with building projects using Phidgets. All the students were able to complete the assignment, undertaking a wide variety of projects, and reported that most of their effort was spent on physically assembling their device rather than working with the software, as was the intent. Some students voluntarily engaged in minor electronics work such as soldering, but most did not. The students were required to demonstrate their creations to each other, with students being impressed by the creativity and quality of work from other students, providing positive social reinforcement. The end conclusion was that packaging physical devices in this manner greatly simplified their programming, and enabled students to create impressive projects. This 'packaging' approach is also potentially applicable to existing devices, allowing them to be repurposed as a peripheral with a simple interface, even if this was not the original intent for the device.

The .NET Gadgeteer platform was developed by Microsoft Research used a similar

idea of standard components which could be used to assemble devices, but focused on programming them directly through C# or Visual Basic (VB) [8]. This placed it as a category 4 device in the previously discussed categorisation. Like Phidgets, the Gadgeteer platform consisted of components with different capabilities, such as sensors, displays and ports, that could be connected together in different ways through a mainboard. The mainboard could then be programmed in Visual Studio on a computer using the .NET Micro Framework through a simple component-oriented interface. The main difference from Phidgets was the ability to use the resulting programmed device independently, rather than needing to remain connected to a computer. The use of the complex Visual Studio development environment and requiring specific setup to work with the Gadgeteer platform made it difficult to use for some students and teachers [23]. However, students also appreciated being able to develop using tools and programming languages used in the professional world and solving useful problems in a way they could physically interact with. Another issue was that the complex setup limited the ability for the Gadgeteer to be used in small classes or as an after school activity. Nevertheless, it was able to achieve similar benefits to other physical device studies regarding improved motivation and relevance [8, 23], however it now appears to be discontinued.

A different approach to a directly programmable device is the 'LilyPad', which is an Arduino-based device designed for use in electronic textiles ('e-textiles') as a means of reaching a broader range of students [3, 4]. The LilyPad which operates as a more traditional category 4 device like other Arduino devices in the discussed categorisation. Recognising that fashion is an area of significance to teenagers and pre-teens as they begin developing an independent identity, the studies explored the development of a device and course specifically focused around combining electronics, computer science and fashion. The resulting concept of e-textiles involved a different paradigm to alternatives such as robotics, as a LEGO-based robot can be easily taken apart and re-assembled, but an accessory or item of clothing is a lot harder to disassemble. While this can impede experimentation in some ways, the permanence of the creation also meant that individuals put a large degree of thought into their projects and could make use of them in their daily lives. In some cases, the artefacts created by the students were praised by their peers outside of the course, leading to greater interest into computing in their social groups as a result of the artefacts' visibility. To help combat the risks of mistakes being harder to correct, prototyping and planning skills were a focus of the course, which are also beneficial skills to emphasise in embedded programming where there can be similar constraints in professional applications. The major issue encountered in the study was the difficulty inexperienced students found programming the LilyPad boards through the Arduino development environment, which uses a language based on C++. The studies

acknowledge that e-textiles may not appeal to everybody, the majority of participants in the studies were female, but stresses the idea that for individuals historically excluded from computer science, meeting them in hobbies or communities they already engage with can be an effective method of demonstrating that they can also engage with computing.

A project which sought to tackle the difficulty of programming Arduino-based microcontrollers, which are prolific in the hobbyist and educational spaces, was 'Modkit', a Scratch-inspired block-based programming environment for Arduino boards [19]. Modkit was inspired by work on programmes with young individuals from underrepresented groups, including women and ethnic minorities, which address issues in their communities through computing. A repeated obstacle encountered in these programmes were the difficulties students had learning the Arduino language, requiring a considerable amount of support. Many of the individuals had existing proficiency with Scratch from previous programmes or courses, so a Scratch-like environment was developed for programming Arduino boards, which still exposed the underlying code for students to inspect. This involved creating some simpler hardware for inexperienced users which still enabled the range of activities necessary for these types of programmes to be effective. The blocks were designed to match the Arduino API in order to both cater to Arduino-experienced students and teach the API to inexperienced students, enabling them to transition towards text-based programming. It also made use of primarily web-based software to simplify setup, with local companion software only being used to enable connectivity with the boards. Although there was not a formal evaluation, based on previous studies it is likely that this would be an effective method of lowering the barrier of entry for inexperienced students, while still teaching them the skills necessary to improve their skills and move to traditional methods.

In 2015, the British Broadcasting Corporation (BBC) announced a project to develop a cheap physical computing device for use in schools and by children to learn computer science, the BBC micro:bit [24]. The micro:bit was distributed freely to schools in the United Kingdom in 2016, with around 800,000 devices delivered. The micro:bit is a device smaller than a credit card, featuring a range of sensors, inputs, an LED matrix display, and and general purpose input/output (GPIO) pins. Delays in the delivery of micro:bit devices impaired the ability of the study to fully evaluate the effects of using the device, but showed that students enjoyed interacting with the tangible devices, found it easy to achieve satisfying results, and found the device accessible even without a programming background [24, 25]. The official micro:bit Python Editor was found simple and easy-to-use by students, and block-based programming was also supported through Microsoft MakeCode for micro:bit. Similar to platforms like the Gadgeteer and Arduino via Modkit, the device was directly programmed rather than used as a peripheral, placing it as a

category 4 device in the discussed categorisation. One concern less present with other approaches was the device's potential limitations, as the Gadgeteer and Arduino platforms had a range of different components available while the micro:bit only had its on board functionality, however that has been mitigated with the release of accessories designed for the micro:bit through its 'edge connector' [9]. The micro:bit includes a lot of functionality on the device and the inclusion of GPIO makes it theoretically as extensible as an Arduino-based device. Uniquely however, the support of the BBC enabled the device to become rapidly prolific, causing it to gain attention outside of the UK, evident by its use in the Irish computer science curriculum as well as in Canada, Croatia, Denmark, Hong Kong, Norway, Uruguay and Singapore, among others [9]. This has been facilitated by the micro:bit's MicroPython interface, block-based programming option and web-based tools, which make the device more accessible to inexperienced users.

2.1.4 Encouraging Diversity

As has been mentioned previously, physical interventions can be a particularly effective method of better engaging female students in computer science [9, 10, 24] and helping to grow their confidence [9, 23, 24]. One reason suggested is the improved relevance to real life applications and problems, which has been identified as an important aspect of motivation in multiple cases [3, 17, 19, 21, 28]. This can take the form of developing alternative pathways that engage female students by integrating with activities or communities they are already engaged in [3, 28], making it easier to address issues of importance to them in their communities [17, 19], or relating the content to technology that is a regular part of their lives [21]. A common element throughout is the need to expand the idea of who a computer scientist can be through presenting a range of different entry routes into and applications of computer science, appealing to a variety of girls and women. Additionally, there has been evidence that interventions utilising the LilyPad [3] and micro:bit [9] have improved the disposition of female students to continue studying computer science, further indicating the efficacy of physical devices in encouraging female students.

A major obstacle cited for underrepresented groups is the lack of encouragement or support they receive in pursuing computer science, which can make it more difficult to engage with computing programmes and courses [3, 17, 28]. When a student lacks external support or encouragement, obstacles to engaging with computer science can be more likely to contribute to frustration and ultimately attrition [17], making it especially important that educational interfaces are not frustrating, appear inviting, and facilitate users solving their problems, rather than just presenting them. User-friendly and 'error proof' design has been effective in Scratch and other block-based programming environments

at reducing frustration and lowering the barrier to entry [15, 16], and similar elements of built-in documentation and helpful feedback, such as in the official micro:bit Python Editor, may achieve these effects beyond block-based programming.

Given the time constraints of the project, it is unlikely that it will be able to produce statistically significant results on whether the approach it follows is specifically effective at engaging or retaining underrepresented groups. Furthermore, factors affecting the inclusion and engagement of ethnic minorities are greatly affected by the contexts of specific countries, requiring substantially more focus, detail and expertise to address. However, the solution implemented could be evaluated further in order to study these factors in more detail over an extended period, which has not been studied substantially at the pre-university level. The inclusion as part of an educational environment currently under study and the use of an already popular device in the project would make such studies easier to conduct, while still fulfilling the criteria proposed to be responsible for these benefits.

2.2 Technical Architecture of Pytch

The Pytch project consists of multiple components stored in separate source code repositories with responsibility for different aspects of the project, including the environment itself and its supporting resources and website. There are two components relevant to the project: the React-based web application ('web app') providing the user interface and the JavaScript-based virtual machine ('VM') providing the modified Python interpreter.

2.2.1 Web App

The web app is built using the popular React library for developing responsive web-based user interfaces using the JavaScript XML (JSX) syntax. Websites using React are often delivered as Single Page Applications (SPAs), consisting of one actual web page rendered by the browser which is modified and updated by a client-side library. SPAs can have a number of benefits, such as allowing more responsive and interactive user experiences, as most of the work is done locally on the client side, meaning interactions do not require a network request to complete before the interface is updated. The web app also makes use of the TypeScript programming language, a superset of JavaScript, adding static typing and enabling improved static code analysis, which can be used to better identify potential problems in the code.

The environment itself is implemented within an `IDE.tsx` component, which composes a code editor, Scratch-style 'stage' and information panel together to form the user

interface. Project-level and other global state is shared between different components through the use of the Easy Peasy library, which is an abstraction on top of the Redux state manager. The logic which connects the user interface to the VM is separated from the React components, handling providing the written code to the interpreter, triggering Python functions based on browser events, and enabling the VM to modify the stage or play sounds. The majority of the modifications to the user interface will come to the information panel, which will contain the new interface for connecting and managing physical devices, and the interconnect with the VM.

2.2.2 Virtual Machine

The VM is based upon the pre-existing Skulpt project, which implements an interpreter largely compatible with Python 3.7 written in JavaScript, along with parts of the Python standard library. Skulpt features a system known as 'suspensions', which allow for asynchronous execution of the interpreted Python code, enabling interactions from Python with asynchronous or event-based browser APIs. Code used by Skulpt can either be written in JavaScript as 'builtins', equivalent to native C functions in the reference CPython runtime, or in Python. Generally in Pytch, where a function would need to interact with JavaScript code or the browser directly, it is implemented as a builtin, and all other functions are implemented in Python.

For its specific functionality, Pytch provides a `pytch` library as part of the standard library bundled with the VM. In particular, the project and actor system used to provide Scratch-like concurrency is implemented in `project.js`, and the existing asynchronous and browser-interacting functionality is implemented in a `syscalls.js` file, both as JavaScript builtins. Asynchronous code for device communication will be implemented in these two files, but the library provided for interfacing with the device can be implemented separately.

2.3 Physical Computing Devices

There are a number of physical computing devices available, so the potential options need to be researched in order to select the optimal device for the project. This section will focus on detailing the options available, while the criteria used for selecting a candidate device will be discussed in section 3.2.

After examining the devices used in studies of previous approaches discussed above, as well as the devices manufactured by major vendors in the hobbyist and educational spaces, 7 devices were identified as candidates. These devices are: the BBC micro:bit

v2, the Raspberry Pi Pico, the Arduino UNO R4 WiFi, the Maker Pi Pico, the Adafruit Circuit Playground Classic, the Oxocard Mini Galaxy, and the Joylabz Makey Makey. This selection represents three major vendors in the space, Raspberry Pi, Arduino and Adafruit, as well as the BBC micro:bit and some relevant smaller vendors. The UNO R4 WiFi was selected to represent the Arduino platform as Arduino manufactures many boards, and the UNO R4 WiFi is the most recent revision of the flagship UNO board, has an affordable price and a strong feature set. The Maker Pi Pico was selected to represent the ecosystem of more fully featured kits built around the Raspberry Pi Pico which aim to make the device more readily and easily useful. A comparison of the devices can be found in Table 2.1.

Other devices that were researched to inform the design of the project, but were unable to be considered as candidates due to their being discontinued, include the SparkFun PicoBoard, the CodeBug, and the Microsoft .NET Gadgeteer platform.

Device	BBC micro:bit V2	Raspberry Pi Pico	Arduino UNO R4 WiFi	Maker Pi Pico	Circuit Playground Classic	Oxocard Mini Galaxy	Makey Makey
Price*	€14.27	€5.84	€20.99	€9.73	€15.81	49 fr.	€47.95
Buttons	3	0	0	3	2	5	6
Display/LEDs	5x5 matrix	1 LED	12x8 matrix	1 LED	10 RGB, 1 red	240x240 RGB	6 LEDs
Connectivity	Serial, WebUSB	USB host	USB host, Serial, WebUSB	USB host	USB host, Serial	Serial	USB HID
GPIO	3 pads	26 pins	20 pins	20 pins	8 pads	26 pins	6 clips, 12 pins
Speaker	Yes	No	No	Buzzer	Buzzer	Yes	No
Microphone	Yes	No	No	No	Yes	No	No
Motion	Yes	No	No	No	Yes	No	No
Temperature	Yes	No	No	No	Yes	No	No

Table 2.1: A comparison of the candidate devices

** Prices sourced without VAT from The Pi Hut as a point of comparison, except for the Oxocard Mini Galaxy and Makey Makey where prices were sourced from their manufacturers*

2.4 Similar Existing Solutions

Currently, there is not an existing solution that implements something entirely similar to the aim of the project, however there are a number of existing solutions which are similar to aspects of the project. MIT's Scratch with the micro:bit extension is the most similar solution, however there remain some significant differences. Other environments primarily focus on programming physical devices directly through block-based or text-based approaches, but these will still be discussed.

2.4.1 Scratch

Scratch's ability to interface with physical devices comes through the use of Scratch Extensions, which enable hardware like LEGO robotics kits, the Makey Makey and BBC micro:bit to be incorporated into Scratch projects. Some of these extensions offer more complete implementations of the underlying hardware's functionality, like the LEGO extensions, however these are primarily mobile robotics platforms and are less suited to a tethered experience. The micro:bit extension does not expose the full functionality of the device, and only supports functionality available on the original micro:bit. The limitations inherent to Scratch could also make it difficult to program components of the micro:bit even if they were supported, such as its radio and speaker.

The project aims to implement all of the functionality available on the micro:bit, only excluding low-level protocols like SPI and UART which are unlikely to be utilised by an inexperienced user. The Pytch environment's use of standard Python also makes interacting with more complex components, like the aforementioned radio and speaker, more feasible, through the use of string manipulation and lists. While the project uses a similar tethered approach to Scratch, it aims to enable far more potential activities by interacting with more of the device's features and facilitating more complex programs in an approachable manner.

2.4.2 micro:bit Editors

There are two main editors for programming the micro:bit directly, the official micro:bit Python Editor and Microsoft MakeCode for micro:bit. The official Python Editor only supports text-based programming in Python through the BBC MicroPython implementation, while MakeCode supports block-based programming and text-based programming in JavaScript or Python through a specific MakeCode micro:bit runtime. In both cases, the editors are designed for writing standalone micro:bit programs which run on the device itself, rather than being components of a broader environment, and the user has to flash

the new program onto the micro:bit in order to run it, or run it on the included simulator.

The project instead incorporates the micro:bit as a component of the Pytch environment, allowing the Pytch environment and a user's already existing Pytch programs to leverage the capabilities of the micro:bit. This allows for faster iteration, because the on-device software does not need to be updated to reflect changes in the program since it is being used as a tethered component, and enables the micro:bit to be easily used in larger projects not possible on the micro:bit alone. The project and these editors are useful for different types of projects, and the API in the project is designed to be similar to BBC MicroPython to facilitate smooth transitions to the official Python Editor and back to Pytch.

2.4.3 Modkit

Modkit was a block-based programming environment based on Scratch for programming Arduino microcontroller boards, which no longer appears to be accessible as of 01/04/2024. Modkit was designed to make fully-functional embedded programming more accessible to inexperienced users through the use of block-based programming, while still allowing the programs to run independently on the microcontroller as a standard Arduino program would. In this way, Modkit was similar to the block-based programming mode of Microsoft MakeCode for micro:bit, but for the broader range of Arduino-based devices.

The original Modkit environment for the Arduino platform no longer appears to be actively maintained or accessible as the download, documentation and shop links for the environment lead to missing pages. The similar Modkit for VEX environment is still accessible and appears to be functional, but its documentation is also no longer available. While Modkit offered a more complete set of functionality available on the supported Arduino boards and peripherals compared to Scratch extensions, it did not incorporate this interface into a broader Scratch-like environment as the project intends to.

2.5 Summary

Computer science education is an area of active and extensive research, with numerous open source projects and commercial products available to cater to different needs and subject areas. In recent years, particular attention has been paid to the area of physical devices and their role in computing education, with the development and availability of various devices and accompanying tools, and the study of those devices and tools. However, many previously studied solutions are either no longer available or do not necessarily meet the needs of many learners and educators. By filling a gap in both research

and tooling, the project aims to continue studying the purported benefits of physical devices through a different approach and broaden the solutions available to students and educators.

Chapter 3

Design

This chapter will detail the requirements and constraints identified for the solution based on the goal and objectives outlined in section 1.3. This chapter will then detail the criteria for selecting the physical computing device to be used and the justification for the device that was chosen. Additionally, this chapter will explore the design process for the software which will run on the device, the interconnect between the browser and the device, and the design of the Python library which exposes the device functionality to the user. Finally, this chapter will explore the design of the sample activity used in the evaluation, as well as the design of the evaluation method.

3.1 Requirements

The requirements can first be defined generally in terms of the project as a whole, and can then be applied more specifically to each of the four main areas of design in the project. These requirements will be informed by a combination of the design principles of Pytch and the research conducted into other projects with similar goals, as well as the underlying theory regarding the benefits of physical approaches.

In order for the project to achieve the goal and objectives outlined in section 1.3, it needs to meet the following requirements.

R1 The solution needs to be feasible for schools to deploy and use without requiring specific training or substantial cost for the school, teachers or students.

R2 The solution should be simple to set up and should work reliably, with straightforward troubleshooting at most, such that an inexperienced user can set up and use the solution with minimal instruction.

R3 The solution should enable the program to interact with the user and the physical world to a significant degree, with a sufficient degree of flexibility to facilitate different activities at varying difficulty levels.

R4 The solution should enable experimental techniques and approaches which are engaging, motivating and rewarding for students, and avoid frustration or boredom.

R5 The efficacy of the solution should be verified and obvious, such that a student, teacher or instructor can understand the additional value provided.

R6 The solution should abide by the design principles and conventions of Pytch, while remaining similar to the native interface of the device, in order to facilitate interactions with Pytch and transitions between Pytch and the native interface.

The first two requirements are important so that the output of the project is usable by the target audience of the project, acknowledging that an effective solution is only useful insofar as the intended user has the capacity to engage with it. The third and fourth requirements are important so that the design and implementation of the solution enables the benefits associated with tangible and hybrid approaches, in line with the research described in section 2.1. The fifth requirement is important so that the benefits of the completed project are sound and verified, so that its intended users are encouraged to adopt the solution. Finally, the sixth requirement is important so that the output of the project works cohesively with the existing functionality of Pytch, and so it can facilitate the transition to more advanced levels of learning in a similar manner to Pytch. The specific applications of these requirements will be referenced in each of the following sections where applicable, with the requirements they are related to indicated in brackets.

3.2 Physical Device Selection

Building upon the requirements for the entire project, the following criteria were identified as being important when selecting a physical device for use in the project:

1. The device selected should be cheap enough so most schools can afford to buy a sufficient number for their class, and most students can afford to buy one for their own use. (R1)

2. The device should require minimal steps to be ready to use and have minimal maintenance requirements, while remaining fully functional. (R2)

3. The device should be reasonably durable, so that it will remain functional through repeated and extended expected use, and will withstand typical wear in a school environment. (R2)

4. The device should have a range of sensors, inputs and outputs readily available without requiring the use of additional components. (R3, R4)

5. The device should be physically extensible and allow direct programming to ensure it is useful beyond its application in the project. (R3)

	micro:bit	**Pi Pico**	**Arduino UNO**	**Maker Pi Pico**	**CP Classic**	**Oxocard**	**Makey Makey**
1	Y	Y	Y	Y	Y	N	N
2	Y	N	N	Y	Y	Y	N
3	Y	N	Y	Y	Y	Y	Y
4	Y	N	N	N	Y	Y	N
5	Y	Y	Y	Y	Y	Y	Y

Table 3.1: An evaluation of how each device fits the criteria

Based on the above criteria, only two devices meet all 5 criteria, the BBC micro:bit V2 and the Adafruit Circuit Playground Classic. While not one of the original criteria, the micro:bit has seen wide deployment at schools in the United Kingdom and is used in the main textbook available for Leaving Certificate Computer Science in Ireland [1], as well as having resources available on the portal for teaching resources maintained by the Irish Department of Education. While not a strict requirement that would prevent the use of a more appropriate device, these are additional benefits to selecting the micro:bit, as schools may already be in possession of these devices, making the output of the project more accessible, and they will have use beyond the output of the project, making them a more attractive investment for schools. The micro:bit has also been the subject of or utilised in prior research, providing useful information about the use of the device in an educational setting and validating it as a suitable choice.

This makes the micro:bit a better choice than the Circuit Playground, especially given that the micro:bit is cheaper when bought in kits of 10 devices and the Circuit Playground lacks some of the micro:bit's features, such as having a radio and a matrix display rather than individual LEDs. This makes the micro:bit the best option as the primary device for the project, with the Circuit Playground being an option for future expansion. Although the V2 revision of the micro:bit was used for the comparison, the original version of the micro:bit also meets all of the criteria, which is another benefit as schools or students may

already be in possession of this device. As a result, the project will also aim to support the original version, but without the functionality only available on the V2 revision.

3.3 Technical Design

The technical design of the project will span three technical components, the on-device software, the browser interconnect and the Python library. Since these components are deeply interrelated, how they will interface with each other must be carefully considered in their design. The design of the Python library will be discussed first, followed by the browser interconnect and finally the on-device software. This follows a user-centred design approach, beginning with the interfaces the user interacts with and then working towards the supporting implementation.

In order to minimise friction for users and create a positive learning experience, it is important for the programming interface in the Python library to be as intuitive and understandable as possible when exposing all of the relevant functionality to the user. The browser interconnect can then be developed working backwards to enable the desired interface in the Python library, while still ensuring that the user interface for managing the devices connected is straightforward to use for an inexperienced user, as it will be the first step for a user who wishes to use a connected device in their program. Finally, the on-device software can be developed to communicate over the interconnect with the Python library in order to enable its functionality. This will then yield a complete design for the technical components of the project.

When considering the design of each component, the following criteria will be used to align the technology with the requirements of the project as a whole:

1. The technical design should expose a 'Pythonic' (idiomatic) interface which follows best practices for the language, is in keeping with the overall style of Pytch, and is similar to the native interface for the device. (R2, R6)

2. The technical design should make it simple for a user to connect and set up a device without any prior knowledge, and should indicate to a user if a device is required for the current program but one is not currently connected. (R2)

3. The technical design should be reliable, automatically recovering from invalid states where possible and providing clear instruction to the user where automatic recovery is not possible. (R2)

4. The technical design should implement the entirety of the functionality available on the device, with the exception of certain functionality that is unlikely to be

useful within an environment like Pytch or for inexperienced users, such as low level communication protocols. (R3, R4)

3.3.1 Python Library

As mentioned while discussing the architecture of Pytch in section 2.2, the functionality specific to the Pytch environment is contained within the `pytch` library provided with the Pytch VM. In a similar manner, the functionality specific to working with the micro:bit will be contained within `pytch.microbit`. Separating the micro:bit specific functionality from the broader Pytch functionality is beneficial for a few reasons. Firstly, it draws a clear distinction between functions which interact with the Pytch environment and which interact with the device, which are able to be imported with `import pytch.microbit as microbit` and used in the format of `microbit.action()`, which is easier to understand for an inexperienced user. Secondly, it allows the user to selectively import the micro:bit functionality based on whether they wish to use it in their project, which avoids unnecessarily polluting the context for the user when the micro:bit is not being used, and has the side benefit of subtly introducing the concept of Python 'modules' containing different functionality to the user. Thirdly, it allows for easier development by separating out the added micro:bit functionality, which avoids cluttering the `pytch` library and reduces the likelihood and severity of merge conflicts between different changes.

Similar to the main library, the functionality exposed by the micro:bit library will be separated into actions, properties and 'hat block' decorators for events. However, unlike the main library, the functions are exposed directly rather than existing as methods on classes. In the context of the main library, accessing functions as methods or properties on classes makes sense, as classes are used to represent the stage and sprites in the Pytch environment, collectively 'actors', and multiple actors will be defined by the user in any given project. For a device like the micro:bit however, it is likely that only one device will be used at a time and students will want to interact with that one device easily from any actor in their program. This means that it makes more sense to expose the functionality directly rather than through a class.

When naming the micro:bit functions, there are different considerations for the actions and for the hat blocks. For the actions, there are few similar functions in the main Pytch library, as the functions primarily deal with the Pytch stage and sound, but these functions can serve as inspiration for some of the related functions for the micro:bit. By drawing on the BBC MicroPython API reference, the function names can also be kept similar to the native environment for programming the micro:bit. One key difference is that in

the MicroPython API, the functions are divided further into different modules, such as `audio` and `display`, which adds an unnecessary level of complexity for our case. Instead, the function names can be made clearer themselves, for example `play_music` rather than `audio.play`. These names stay close to the underlying MicroPython function names but avoid the additional layer of modules. For the hat blocks, there is no similar concept in the MicroPython API, so the main Pytch library is the sole influence. Pytch hat blocks are named in a similar way to the hat blocks in Scratch, they begin with 'when' followed by a description of the action, for example `when_key_pressed`. This can be replicated closely, for example `when_button_pressed`.

Interfaces for all of the core micro:bit functionality will be needed, including its sensors, speaker, display, radio and GPIO. The MicroPython API provides some functionality tied to specific peripherals not included on the board, such as NeoPixel LEDs, and for low level protocols used to connect to other devices, like SPI and UART. The former functionality is not within the scope of the project, as it pertains to components that the user may or may not have access to and are not a part of the selected device. The latter functionality is also not within the scope of the project, as these protocols are specifically used for communicating with other devices and are likely to be too complicated for an inexperienced user to make use of. Implementing them could be a future improvement if a use case is identified.

3.3.2 Browser Interconnect

Based on the desired interface for the Python library and with reference to the second and third technical design criteria especially, using USB as the basis for the browser interconnect is the best option for achieving the project goal and objectives. Although the micro:bit has a radio which is capable of communicating over Bluetooth when the device is flashed with a Bluetooth software stack, the use of Bluetooth as the connectivity mechanism in the project has three main disadvantages. Firstly, the radio functionality on the micro:bit cannot be used when Bluetooth is being used, which prevents one of the micro:bit's key features from being used in Pytch, preventing many potential activities. Secondly, pairing a device via Bluetooth requires more steps compared to plugging a device in with a cable and has the potential to pair with the incorrect device, especially in a classroom environment with many devices, which would introduce more friction, and potentially user frustration. Thirdly, since Pytch is primarily used in classroom environments, having a whole class of students using Bluetooth devices at once may result in interference and cause unstable connectivity, which could be exacerbated in the presence of other Bluetooth devices, such as smartphones. Together, these potential issues

make it clear that a wired method is the best option for connecting the device reliably, even if it may restrict the movement of the device and limit some activities which would require movement, as the tethered design of the project limits the possibility of such activities regardless but is necessary for integrating the device into an environment rather than using it by itself.

There are two main options for connecting to a USB device from a website, one is to use companion software as an intermediary between the device and the website, and the other is to use the draft WebUSB API, which is currently implemented in the Chromium project and the various browsers based upon it, such as Google Chrome, Microsoft Edge, Opera and Brave. The use of companion software has been adopted in applications like Scratch and Arduino Cloud, while WebUSB has been adopted in applications like the official BBC micro:bit Python Editor and Microsoft MakeCode. Not all devices are capable of being fully programmed via WebUSB, which may necessitate the use of companion software, however the micro:bit features the Arm Mbed DAPLink software running on its USB interface, which enables communication and device flashing over WebUSB. While companion software can offer more potential for interacting with different devices beyond the micro:bit and can support a broader range of browsers beyond Chromium-based browsers, it has four drawbacks which are significant with respect to the requirements.

Firstly, companion software needs to be installed, which is an obstacle to use for an inexperienced user and often requires administrative privileges, which an inexperienced user may not have or may be entirely disabled on a school-administered device. Secondly, companion software may have platform-specific requirements, so separate versions may need to be developed and maintained for the different operating systems used in schools which adds additional development and maintenance work and may not be possible for ChromeOS, which is widely used in schools due to the popularity of Chromebooks. Thirdly, companion software may need to be updated to support more devices in the future or to fix bugs which adds additional work on the user's end when an update is released, making it more complicated for them to use. Fourthly, the main way of communicating with a device would be to use a standard protocol available in the browser, like HTTP or WebSockets, but this may also open up opportunities for malicious actors to try and exploit such software in order to gain access to or damage a user's device, which is less of a concern when relying on well-tested APIs implemented by the browser vendor. These four drawbacks make companion software a less appealing option, and there are few drawbacks to using WebUSB with the micro:bit, aside from requiring the use of a specific browser which is less of an obstacle than only being able to support specific operating systems. The use of WebUSB may be a barrier to adopting other devices in the future,

but other desired devices may also be able to be programmed to support WebUSB, and using companion software or other approaches for other devices in addition to using WebUSB for the micro:bit would still be viable. These factors make WebUSB the preferable choice for the project.

3.3.3 On-Device Software

There is one officially supported language for programming the micro:bit, Python, through the BBC's implementation of MicroPython. MicroPython is a complete implementation of a Python 3 compiler and runtime, along with some of the core modules in the Python standard library and some microcontroller specific modules. The BBC implementation extends MicroPython with micro:bit specific modules to make leveraging the sensors, inputs and outputs on the device more straightforward. Microsoft also provides a method of programming the micro:bit via Python through their MakeCode for micro:bit environment, which allows programming via blocks, JavaScript or Python. The MakeCode version of Python does not share the same API as MicroPython and has additional constraints in order to enable MakeCode to freely swap between block-based programming and text-based programming with JavaScript or Python. These constraints and the complexity involved with compiling MakeCode programs locally make it less ideal for developing the on-device software than BBC MicroPython.

Another option for programming the micro:bit is through C++ using the Arduino IDE and the libraries maintained by Adafruit, a third-party electronics manufacturer. These libraries are not officially maintained by the Micro:bit Educational Foundation, the organisation which oversees the development of the micro:bit, and primarily target the original revision of the micro:bit rather than the newer V2 revision. As a result, although leveraging the Arduino IDE and libraries could make the on-device software more easily portable to other devices, the lack of official and up-to-date support mean that it is not an ideal option for the project. Since the BBC MicroPython implementation is open-source, it is possible to examine the code for the implementation and develop the on-device software using C in a similar manner, however the original revision and the V2 revision of the micro:bit use different device abstraction layers (DALs), meaning that two versions of the software would be needed and this process would be substantially more involved for limited benefit. As a result, using the officially supported MicroPython implementation to write the on-device software in Python is the most suitable approach and also provides the ability for users to easily modify the on-device software to their own requirements.

Within the on-device software, there are two essential functions that need to be per-

formed. It needs to communicate sensor-based events through the interconnect to trigger the hat blocks in the Python library and it needs to respond to commands sent through the interconnect from the Python library to perform actions, such as displaying an image, or provide sensor values, such as acceleration. This can be achieved through a simple event loop, which checks for incoming commands from Pytch over the serial interface, processes the commands and responds to them, then checks the sensors for any new events and sends the events to Pytch. This order has been chosen intentionally since events triggered by the micro:bit may themselves cause further commands to be sent, so processing the current backlog of commands before triggering any new events makes sense. In practice, most commands should execute quickly as they will be performing small changes to components on the board, and the loop will likely be running multiple times per second, meaning there should be minimal latency for processing events.

3.4 Sample Activity

Drawing on the learning theories of constructivism and bricolage alongside the insights from previous physical computing interventions in computer science education explored in 2.1, the following criteria for a sample activity can be identified:

1. The sample activity should be feasible to accomplish under time constraints typical for a single class in secondary school and remain engaging to a student throughout. (R1)

2. The sample activity should include clear guidance and supplemental resources for an instructor without prior knowledge of the solution, so that the activity can be accomplished by students with a range of abilities without excessive frustration which may be demotivating. (R1, R2)

3. The sample activity should be of an appropriate difficulty level for the intended students, being sufficiently challenging to engage students without being overly complex and causing frustration. (R3, R4)

4. The sample activity should demonstrate the capabilities of the device so that a student is able to conceive of further projects leveraging the additional capabilities. (R3)

5. The sample activity should cater to the interests of the users targeted in order to initially capture their attention and establish relevance to their own lives. (R4)

6. The sample activity should involve a physical or tactile component leveraging the device, in order to achieve the benefits associated with a tangible approach. (R4)

7. The sample activity should interact with the Pytch environment in some way in order to explore the benefits of the hybrid approach rather than either component in isolation. (R4, R5, R6)

Initially, the development of multiple sample activities was considered, however it was decided to focus on developing a single activity for a number of reasons. Firstly, the time constraints imposed on the project made multiple interventions with the same group unlikely, meaning that any intervention conducted would be with unfamiliar users and would need to introduce the components, preventing the assessment of multiple linked sample activities. Secondly, in order to get the most useful results from the limited interventions, it was important to have a very high quality sample activity to assess the project through to avoid negatively influencing the results due to a poorly constructed activity, and focusing effort on a single activity would yield a much higher quality.

With these criteria in mind, it was decided for the sample activity to be a game. Games have been used frequently in previous research, both in computer science education and education more broadly, as a method for capturing the interest of students, especially younger or less technically experienced students [11, 20]. Additionally, games as a project lend themselves well to a bricolage approach, as an incomplete game can be provided for a user to complete and expand upon, helping to develop an understanding of the internals of the game through exploration and stimulating creativity by allowing users to alter the game to their own liking. Finally, developing a game which the user enjoys results in a level of satisfaction, since it can also be shared with and appreciated by others who are not interested in computer science, as well as being an enjoyable part of the experience for the user itself. When designing such an activity, it is important to select a game that can make use of the functionality available from both the micro:bit and the Pytch environment rather than making sole use of the micro:bit, in order to capitalise on the specific benefits of a hybrid approach. The most straightforward method of incorporating both elements is for the micro:bit to be used as a controller for a game that is primarily experienced on the stage of the Pytch environment. However, just using the buttons on the micro:bit as input for a game does not provide much additional value compared to using the keys on the keyboard, so a more novel approach to input is required alongside making some other use of the micro:bit as well.

There are three main sensors on the micro:bit that lend themselves to being used for user-driven input: the microphone, the accelerometer and the light sensor. The microphone is capable of being activated by claps and other loud noises, which could be used

for rhythm or reaction based games, however this input method may run into difficulties in a classroom environment where the interventions would take place. The light sensor can be affected by covering or uncovering the micro:bit's display which could work similarly to a button, however this input method has more potential to be inconsistent and frustrating for a user depending on the environment, and only having a single input limits the options for the activity. The accelerometer can measure acceleration along three axes and the MicroPython API provides basic gesture recognition for tilting and shaking the micro:bit, providing multiple options for control and presenting a solid option for controlling the game. Accelerometer-based controls are relatively common in video games, especially in party and driving games, and these controls can be intuitive to a user as they often mimic interactions with the physical objects the controls represent, such as tilting a controller to steer a car. A few potential games can be considered for implementation in the sample activity using the accelerometer. Broadly, the types of gameplay this allows can be divided into some form of movement control and some form of action control. Tilting or moving the device can be clearly mapped to motion in the relevant axis, and shaking the device can be clearly mapped to an action of some sort.

'Platformer' style games are often one of the first examples that come to mind for simple movement-based games, however such games often involve the creation of levels and enemies, which may be too complex for the intervention. 'Party games' also often benefit from simple controls rather than being disadvantaged by them, relying on creative game design such as translating the controller being shaken to throwing a dart or having a player avoid an obstacle briefly, however such a game may be difficult for the user to adapt to their own tastes due to the specificity. Driving games generally only require two basic controls in order to steer in each direction, optionally having acceleration and braking controls for more complex games. While a complex driving game is not realistically achievable within the criteria established, a simple 'endless runner' game similar vein to popular mobile games like 'Subway Surfers' is possible, and is a game concept that the users targeted by the project are likely to be familiar with. In this style of game, the player has to avoid obstacles for as long as possible and optionally collect items. The game can be rendered primarily on the stage of the Pytch environment while the player character can be controlled through gestures with the micro:bit. Additionally, the game can be augmented by displaying the score achieved by the player on the micro:bit's display and by playing sound effects through the micro:bit's speaker. Such an activity should be sufficiently engaging without being excessively complicated, and allow for modification by incorporating additional components or altering the values of the main variables. Some form of 'skeleton' or template which the students can explore and modify through a bricolage approach could be an effective way of creating a game-based activity.

The radio available on the micro:bit presents the opportunity for activities to become more social or interpersonal since the radio enables simple communication between multiple devices. The inclusion of the radio in the sample activity was considered, allowing for users to in some way interact with each other across their devices, however this was ultimately deemed too complex to incorporate into a single intervention given the identified criteria for the sample activity. The radio messages are simple strings, so some form of message structure would need to be developed, including encoding and decoding the values in the messages, which is a significant level of complexity to add to the activity. Such an exercise could be well suited to a follow up intervention which builds upon the results of the first, but the previously discussed time constraints of the project would not allow for this.

3.5 Evaluation Method

The project will need to incorporate interventions with its intended audience in order to be able to effectively assess whether it has been successful in achieving its goal. A comprehensive experimental evaluation will involve conducting a study using the output of the project, delivered in the form of the sample activity discussed previously and collecting data to assess the effects. While it would be ideal to conduct a series of interventions to examine the long running effects of the output, potentially with different groups and approaches to explore different aspects of the project, the time constraints imposed prevent an extensive study from taking place. Accordingly, the project will only be able to be evaluated through a small number of interventions, meaning only indicative results will be able to be established, suggesting whether the project achieved its aims and whether the approach merits further study. In order to establish this, it would be valuable to collect both quantitative and qualitative data regarding the activity. Quantitatively, data regarding student motivation and learning outcomes can be collected through the use of validated instruments in a questionnaire conducted before and after the activity. Qualitatively, data regarding any problems encountered during the activity and student attitudes towards the activity can be collected through the use of open ended questions in the post-activity questionnaire and observations recorded by the researcher during the course of the activity. The combination of this data should enable the drawing of some tentative conclusions as to the efficacy of the approach regarding student motivation and learning outcomes.

In constructing the two questionnaires, there were three validated instruments considered, all of which were calibrated against a 5 point Likert scale, which will also be used in both questionnaires. The three validated instruments were the Science Motivation Ques-

tionnaire II (SMQ-II), the Instructional Materials Motivation Survey (IMMS), and the Computer Programming Self-Efficacy Scale (CPSES). The SMQ-II deals primarily with motivational questions regarding the study of scientific subjects, while the IMMS deals with a combination of motivation and learning outcomes based upon a specific lesson, and the CPSES deals with an individual's ability and confidence in their ability to successfully accomplish common computer science tasks. Given the focuses of each instrument, questions from the SMQ-II and CPSES were considered for both the pre-activity and post-activity questionnaires in order to identify potential differences in attitude towards computer science and confidence in students' abilities, whereas the questions from the IMMS were only considered for the post-activity questionnaire in order to assess the attitudes towards and effects of the sample activity. The researcher can also note whether a student struggled with or failed to complete an exercise to supplement the learning outcome information provided by the questionnaires. Given the participants in the study will be minors, it is also important that the questionnaire is conducted in a manner such that no personal information about the participants is collected, but a participant's responses to each questionnaire can be connected in order to observe changes in their attitudes and motivation. Once collected, the quantitative and qualitative data can be examined and used to identify strengths and weaknesses of the sample activity and broader project output, as well as whether there were any notable changes in student motivation and whether the intended learning outcomes were achieved. This analysis can form the basis for drawing preliminary conclusions about the implementation and approach.

3.6 Summary

The design of the project involved a range of technical and non-technical components, which had to be carefully considered in relation to each other and the previous research explored in section 2.1. Given the constraints on the project limiting the degree to which interventions were able to be conducted, an emphasis was placed on ensuring the completeness of the technical implementation and its alignment with the established theory, as well as attempting to ensure that the interventions that would be run were of a high quality. The use of overall project criteria which were applied to each component helped to guide the project design and ensure it aligned with the project goal.

Chapter 4

Implementation

This chapter will detail the initial implementation of and subsequent refinements to the components of the project, following on from the description of their design in chapter 3. This chapter will first examine the implementation of each of the three technical elements, then move on to discuss how the sample activity was implemented, with specific reference to the planned delivery and the supplemental materials developed. Finally the chapter will discuss the plan for how the experimental evaluation will take place and the questions used in the questionnaires.

4.1 On-Device Software

As explained in section 3.3, the ideal language for the implementation of the software to run on the micro:bit is Python via the BBC MicroPython implementation. By writing the software using MicroPython it is possible to target both the original and V2 revisions of the micro:bit at once, and the software can leverage the provided functions rather than having to interface with the hardware directly, greatly simplifying the implementation. In the early stages of experimentation, the official micro:bit Python editor was used to test the different functionality as it has a micro:bit simulator, which was useful before physical micro:bit devices arrived, and contains a helpful API reference for the BBC MicroPython API. Later in the process, the software was developed in a standard code editor and was compiled via the command-line to be flashed onto the device.

The first implementation detail to address is how the software on the micro:bit will communicate with a computer. Since the use of a standard web format like JSON would add significant overhead to each message and a JSON module is not available in MicroPython, meaning one would need to be implemented from scratch, using a simple custom message format was the best option. Given the messages are relatively simple, only needing to consist of some keyword with a small number of parameters, the format can consist

of newline-delimited messages with a vertical bar character to delimit between the keyword and each parameter. Messages sent by the browser will always be 'commands' to perform an action or query a variable and messages sent by the micro:bit will always be 'events' driven by the sensors or responses to a command. Commands and events need to have unique keywords respectively, which do not necessarily have to match the names used in the Python library, but should be understandable to facilitate future development and modification by users.

After function and global variable definitions, an event loop runs continuously, divided into two phases. On each iteration, the loop first checks for any pending data on the serial port, continuously reading it into the message buffer until no more data is available. If the buffer contains a complete message, meaning that it is terminated by a newline, then the message is parsed into a command and its parameters, and the function corresponding to the command is invoked with the parameters. This process repeats for each complete message in the buffer. Once there is no more pending data, the loop moves on to checking for events which should be communicated to Pytch. These tests take the form of reading from a sensor, comparing the value to the last known value, and sending an event over the serial port if there has been a change. Once all the event checks have been completed, the loop repeats. In order to make the software easy to understand, each command is implemented in a single function, with the mapping from the command keyword to the corresponding function being stored in a dictionary so that the function can easily be looked up in the command processing loop. These functions take the parameters provided as a list and perform any parsing or validation necessary for their functionality, before executing the corresponding functions in the MicroPython API. In many cases, these functions are one-to-one mappings of MicroPython API functions, but in some cases they are more complex in order to provide a simpler interface to the user, such as with the radio commands.

The `var` command for querying sensor states is a special case, where the corresponding function checks if the provided variable name is valid and returns the relevant sensor values if it is, while most commands do not have data to return. All commands return an acknowledgement even if they do not have data to return, so that the interconnect knows when the command has been successfully actioned. If an error occurs, the command processing loop will catch it and send the error message back to the interconnect instead, so that it can be surfaced in the Pytch environment. Another special case are the `hello` and `identify` commands, which are not invoked by the user via the Python library, but are invoked by the interconnect. The `hello` command can be used to query information about the device itself to display in the interface, such as the firmware revision, and the `identify` command displays an image on the micro:bit's display, with the purpose of

indicating the connected device. The radio event check is also a special case since it is only executed once per loop, to avoid the entire event loop blocking on a large number of incoming radio messages and having the device become unresponsive, which could cause user frustration. Since the event loop is likely to complete quickly in most cases, this should only add a small amount of latency to the radio communication, which should not be noticeable to the user in most cases relative to the latency involved in wireless communication.

In order for the software to be able to be executed on the micro:bit, it needs to be bundled with the MicroPython runtime in the micro:bit binary format and flashed to the board. The Micro:bit Educational Foundation provides a `microbit-fs` library which can be used to manipulate the filesystem used by MicroPython and modify the MicroPython binaries to include a pre-made filesystem. The `microbit-fs` library itself uses the `microbit-universal-hex` library, which is used for creating a single binary that can run on both the original revision and the V2 revision of the micro:bit by bundling binaries for both. The created build script is designed to take the MicroPython binaries files for both revisions and the on-board software, and bundle them into a single binary which can be flashed onto the micro:bit for use with Pytch. First, the latest MicroPython release binaries are retrieved from the corresponding GitHub repositories for the original revision and the V2 revision, which are used to prepare a new MicroPython binary with the `microbit-fs` library. The Python file for the on-board software is then read into a buffer and is used to create a new `main.py` entry in the filesystem for the micro:bit to run on startup. Finally, a universal binary is generated and written out to disk for distribution. This provides a one command method of building a binary file, which can later be integrated into the Pytch build process so that the software can be hosted on the website for download by users.

4.2 Browser Interconnect

The interconnect between the micro:bit and the browser was implemented in the web app component of Pytch described in section 2.2. This component is where all of the user interface for the Pytch environment is developed and also handles integration between the Pytch VM and the browser, such as for data storage, graphical rendering and user input. This logic is approximately divided into three layers, the actual interface implemented in React, the state management system leveraging the Easy Peasy library, and the connection to the VM in standard TypeScript. Each layer will require changes to facilitate the interconnect. The user interface will require new elements for connecting and managing devices, the connection to the VM will need to handle the actual connectivity with the

devices and passing messages to and from the VM, and the state management system will need to bridge the interconnect and the user interface so the interface can be updated automatically by React's reactivity system.

The code implementing the connection to the VM is located in the `skulpt-connection` subdirectory, consisting of a number of modules to handle asset management, keyboard input, graphical rendering, playing sound, building the project and other functions which need to interface directly with the VM. The code specific to communicating with and managing the micro:bit will go into its own module, `device-manager.ts`. The new device manager module will primarily consist of two main classes, a `DeviceManager` class which is responsible for managing all of the connected devices, and a `MicroBitDevice` class which handles communication with a specific connected micro:bit device. The logic has been divided in this way for two reasons. Firstly, it allows for multiple devices to be connected at once and swapped between, which may not occur frequently in typical use but was very helpful for debugging. Secondly, it keeps most of the micro:bit specific code separate from the general connectivity code, making it easier for additional devices to be added in the future.

Since the `DeviceManager` class is in charge of managing the connected devices, only one instance of it is required and it is exposed through a singleton pattern. Having multiple instances of this class may result in race conditions for each instance to claim devices, since devices connected via WebUSB can only be claimed in one place at a time, so having it instantiated in and accessed from one place helps to avoid such issues. When instantiated, the class first checks if WebUSB is supported in the browser so a message can be displayed if it is not, and registers necessary event listeners if it is. Firstly, the class registers event listeners for the WebUSB `connect` and `disconnect` handlers, so that it can attempt to register previously paired devices when they are connected and clean up after devices once they are disconnected. Secondly, the class registers an event handler for the `beforeunload` window event so that it can cleanly disconnect all connected devices, since not disconnecting the micro:bit properly via DAPLink can put the device in an unresponsive state, preventing it from being used again until it is physically disconnected and reconnected. Once the handlers are registered, the class attempts to enumerate any currently connected devices that have been previously paired and register them if any are present.

The registration process consists of a few steps and requires a WebUSB `USBDevice` to be passed into it. First, the process checks to see if the device has already been registered previously, in which case it does not attempt to set it up again to avoid conflicts or race conditions. Then the process instantiates a new `MicroBitDevice` with the ÚSBDevice', registers event listeners on it to handle status changes, and adds the device to its list of

devices. Finally, the process calls the `MicroBitDevice`'s `setup` method to handle setting up the individual device. The registration process can be instantiated in three ways, the two methods previously mentioned, through the `connect` listener and initial device enumeration, and through a user-driven pairing process. The pairing process uses the `requestDevice` function for WebUSB, with a filter limiting it to the micro:bit's USB vendor ID and product ID to prevent the user from selecting an incorrect device. The `requestDevice` function is gated by 'transient activation', meaning that the user has to have performed an interaction such as clicking a button within a certain time window, which is done in order to prevent abuse of this and other powerful APIs, such as the clipboard and payment request APIs. If a device is selected, then it is passed into the `registerDevice` function.

As discussed in section 3.3, only one connected device can be active and used by the current program at a time. The `DeviceManager` exposes this as a read-only `activeDevice` property, which corresponds to either undefined or one of its registered devices. During the device registration process, if there is not currently an active device set when setting up a device succeeds, that device is automatically set as the active device. This means that a user does not usually need to worry about setting an active device since the first available device will automatically become active, and most users will only be using a single device at a time. If the user does have multiple devices connected they can manually set a different device as active. If the currently active device is disconnected somehow then the active device will be unset, and the user will have to either select another device manually or connect a new device which will automatically be set as active.

The `MicroBitDevice` class is instantiated once per device by the `DeviceManager` class, so the `USBDevice` obtained by the `DeviceManager` is passed in during instantiation. The `USBDevice` is first wrapped in the `WebUSB` transport provided by the DAP.js library, and a `DAPLink` client is then instantiated using that transport in serial wire debug (SWD) mode. The actual setup of the device is asynchronous, so is delegated to a dedicated `setup` method, which needs to be called before the device is to be used. This involves calling the `connect` method on the DAPLink client, setting up the serial interface and performing the specific initialisation for Pytch. The initialisation involves executing the special `hello` command discussed previously in order to test that the device is working and to retrieve information about the connected device, such as the hardware and on-device software revisions. Since the serial interface can take some time to actually be initialised correctly by the DAP.js library, which was discovered in early implementations of the interconnect, the function attempts to execute the `hello` command up to 5 times before failing. If a valid response is returned then the information is used to populate details about the device and it is declared ready. If an error occurs at any point during

the setup process the device enters a failure state depending on the type of failure, such as the device being already in use elsewhere or being in an errored state.

When setting up the serial interface, the setup process registers a listener for new serial data which parses the data in a similar manner to the on-device software. It adds the data to the buffer and if there are newlines present the buffer is split and the messages are parsed. The event keyword is then used in a switch statement to trigger the correct event. In most cases, the event has some basic validation applied, and is then added to the undrained event queue in a similar manner to how keyboard events were already handled in Pytch. For the `message` event corresponding to radio messages, the message provided is also pushed to a pending message queue. The `ok` and `err` events are specifically for responding to commands and indicate either the successful completion of the command along with any return values or the failure of the command with the error that was raised by resolving the corresponding promise from the in-flight command queue. The `hello` event is used to update the device's information as previously described. The `MicroBitDevice` class also provides a few methods for interacting with the device, such as cleanly disconnecting the device, unpairing the device, executing the special `identify` command discussed previously, resetting the state of the device, stopping playing music, and sending commands. When a command is sent, it returns a new promise, with the promise's resolve function added to the in-flight command queue mentioned previously.

The `build.ts` module is also important to consider, as it performs various initialisation tasks for the other modules as part of the build process for a user's program when they attempt to run it. Since the Pytch VM cannot access the state or internals of the web app, any modules that the VM needs to be able to interact with must be exposed to it in some way. To achieve this for the `device-manager.ts` module, a `get_active_device` function is passed through to the VM which returns the currently active `MicroBitDevice` instance, if there is one. This will be necessary for sending commands and processing events, which will be discussed in section 4.3. Additionally, when the user restarts their program or runs a different program, they will expect their device to be returned to an initial or 'clean' state and not to be affected by the previous program. To enable this, the `reset` method on the active device is executed during the build process if there is one, which will empty the queues, clear the display and stop any playing music before the newly built program begins to run.

The user interface should respond to various changes in the state of connected devices, so this information needs to be shared with the interface, preferably via the state management system as this information may need to be used in multiple places. The two important pieces of information for the interface to be able to access to display all the information necessary for the user. The first is the list of connected devices, including

their basic information, and the second is which of the devices is active, if any. To hold this information, a new `devices` store is added to the Easy Peasy state model, which holds the list of devices and the active device, and actions to update both pieces of state. This state can then be used in the interface through the `useStoreState` hook provided by Easy Peasy, and it can be updated from the `device-manager.ts` module by using the `getActions` function and calling the corresponding action to update the state when necessary. The interface of the Pytch environment consists of three areas, the code editor, the stage and the info panel. The info panel holds a number of panes for asset management, program output, errors and tutorial instructions if one is being followed, so is the natural location to add the interface to manage devices as a new 'Devices' pane. First of all, if the user's browser does not support WebUSB, then a message informing the user that they will not be able to use devices in their current browser is displayed, along with recommendations for browsers which do support WebUSB. If the browser does support WebUSB then a list of currently connected devices is displayed, along with a button for pairing a new device at the bottom of the list which triggers the pairing process described previously. A connected device can be in one of three interface states. If the device is still in the process of initialising, then the device type is displayed with a spinner, until the device enters either the ready state or a failure state. If the device enters a failure state, then the type of failure state is displayed next to the device type with a hint on how to resolve the issue displayed alongside. If the device enters the ready state, then the revision of the device is displayed, along with an 'Active' badge if it is the current active device, along with four buttons on the right-hand side, 'Set Active', 'Identify', 'Disconnect' and 'Forget', which perform the corresponding actions on the device.

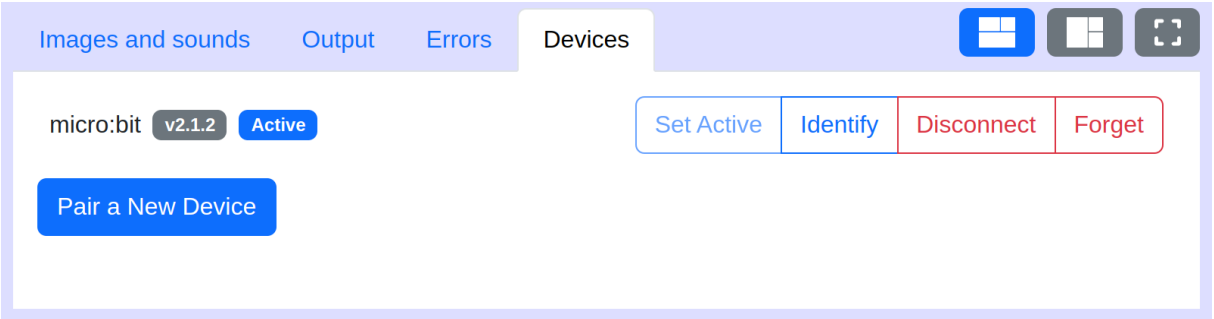

Figure 4.1: The added pane to manage connected devices in the Pytch interface

Additionally, if the user has the `pytch.microbit` library imported but does not have an active device, they will be shown a warning message in the Devices pane, and if they do have an active device but have not imported the `pytch.microbit` library they will be shown a prompt to add the import to their code. This check works by adding

a computed `imports` property to the `project` store in the state model which extracts the imports from the user's code, as well as an `addImport` action which adds an import statement above the first non-import line in the code. The prompt adds the import as `import pytch.microbit as microbit`, which is the recommended way to import the micro:bit library, but the import detection will respect other valid forms of the import to allow for a user to use their own preferred import method. This feature was added to avoid confusion about why micro:bit functions were not working correctly, since a user will need to navigate to the Devices pane to set up a device initially and should see this prompt. The code editor also includes a toggleable help pane, which is divided into the same categories as the blocks in Scratch, and provides documentation about the Pytch API. The help pane shows which Scratch blocks correspond to which Pytch functions along with documentation and code examples, as well as documenting Pytch functions which do not have corresponding Scratch blocks. The help pane is important for facilitating exploration, as it makes it easier for a user to find a piece of functionality they are looking for to solve a certain problem without having to consult external resources or documentation. This is central to the experimental approach to learning, so it is important that the micro:bit functions are also documented here. Since there is an official micro:bit extension for Scratch, the colour coding can be maintained and corresponding Scratch blocks can be displayed for the micro:bit functions where they exist, to ease the transition for users who have used the micro:bit with Scratch and to help illustrate their usage to users familiar with Scratch even if they have not used the micro:bit extension.

4.3 Python Library

The Python library exposing the micro:bit functionality was implemented in the Pytch virtual machine, as described in section 2.2. The VM is responsible for actually running the user-written code and for exposing the Python interface to the various aspects of the Pytch environment, like the stage, input and sound. In order for the micro:bit library to function properly, some changes were required to the virtual machine itself, which involved adding system calls to interact with the `MicroBitDevice` class from the browser interconnect, modifying the Pytch suspension and modifying the event system to support the micro:bit events.

In order for the Python library to be able to communicate with the micro:bit it needs to be able to send commands through and receive events from the browser interconnect. Sending commands can be done by the introduction of a new builtin function which takes the command name and a list of string parameters and triggers a new `microbit-send` 'syscall' with these parameters. The syscalls are a special type of suspension used

by Pytch when interfacing with asynchronous aspects of the browser, enabling custom threading logic. When this syscall is executed on the next frame, it will get the active `MicroBitDevice`, raising an error if one is not connected, and use the `send` method to send the command. The state of the thread is set to a new state, `AWAITING_MICROBIT`, to trigger specific resumption logic, and the promise returned by `send` has a callback attached to store the result in the thread. The specific resumption logic ensures the thread is only resumed once the promise has resolved and the returned value is stored in the thread. Then when the thread resumes, it is tested whether the result was an error, in which case an error is raised, otherwise the value is returned to the caller.

The Pytch VM allows for event-based programming through decorator functions that mimic Scratch's 'hat blocks', adding metadata to the method they are used on about the events the method should be called for. When the actor, such as a sprite or the stage, is instantiated, it checks each method defined on it for this metadata, and if it is present the function is added to the list of handlers for each event the metadata indicates it should be called for. Then each frame the project system checks for events in the queues corresponding to the different event types. If there are pending events of a given type, the system will instantiate a new thread group for each event and request each actor to create a new thread for each handler that should be called for the event, adding them to the newly created thread group. In order to facilitate micro:bit events, additional conditions and checks had to be added at each step of this process.

With these components added, the Python interface can be provided by the micro:bit library on top of the lower level builtins. As discussed in section 3.3, the functionality exposed is generally separated into actions or properties and hat block decorators. The main Pytch library splits the decorators into their own `hat_blocks.py` file, with the actor actions and properties implemented in `actor.py`. To continue this pattern, the micro:bit decorators will be implemented in a `hat_blocks.py` file within the `microbit` subdirectory, and the actions and properties will be implemented in a `device.py` file. The user-facing functions and variables from each will then be exported in a `__init__.py` file to have `pytch.microbit` function as a Python module. The functions map the underlying device commands described previously into more user-friendly function names and perform validation on their arguments to raise helpful errors before the command is ever sent to the micro:bit. In order to access the sensor states on the micro:bit, there is also a `Device` Python class which implements a series of Python properties, equivalent to getters in some other languages, to fetch their values using the previously discussed `var` command.

In the cases of acceleration and the buttons, they are converted to extended NamedTuples rather than just being returned directly to make them easier to use. The NamedTuples map the parameters to their names, mapping the acceleration values to `x`, `y` and `z`,

and the button values to `a`, `b` and `logo`. The acceleration tuple also contains a computed `magnitude` field, and the buttons tuple contains a computed `any` field. Additionally, an `Image` class is provided to make representing images easier. The micro:bit represents images for the display as a 29 character long string, with 5 sequences of 5 digits representing the brightness values of each pixel in each row, separated by colons. This can make it difficult to visualise the image being created, so the `Image` class instead takes 5 lists of 5 digits, so that the code can be formatted in such a way as to make the displayed Image more apparent. This makes custom image creation much simpler for the user, which is valuable for encouraging the type of experimentation and exploration desired in the project and avoiding user frustration. The `Image` class also enables more user-friendly feedback on what aspect of the created image is invalid, rather than just that the image string as a whole is invalid, helping to facilitate learning by experimentation.

The micro:bit decorators are mostly relatively straightforward, taking a specifier for the event like a button or gesture, ensuring the specifier is valid for that event type with similar validation to the actions, and adding the corresponding metadata to the method it was called on. However, the radio message event is more complex as a user may want to accept dynamic messages, such as may be required for sending chat messages, sensor readings, variable states or other dynamic messages. With the current event system in Pytch, events cannot have parameters. This works fine for most cases, as the majority of events are based on specific fixed criteria, like a certain key being pressed on the keyboard, or like a specific button being pressed in the case of the micro:bit. For that reason, the handlers for radio messages need some way to have access to the message that was received. While it may have been possible to modify the Pytch event system to support parameterised events, this would have required significant changes to the existing code which may have introduced other problems and would have posed an issue to the output of the project being merged into the main Pytch project in the future. Instead, the decorator for the message received event was altered to wrap the method it was being used on rather than only adding the metadata. The wrapping function uses an additional builtin function to retrieve the oldest pending message from the active `MicroBitDevice` and pass this to the wrapped function. This approach worked correctly since decorators can replace the function they are being called on, and so the wrapped function existed as a method on the class and was registered correctly as an event handler.

4.4 Sample Activity

Building on the activity concept described in section 3.4, a full sample activity and lesson was developed. First, the desired end result of the activity was implemented to ensure it

was feasible and not overly challenging. Then, once the feasibility was established, the way the activity would be delivered through a lesson was planned out. After the lesson was planned out, some images were created for the activity to replace the placeholder assets that were used for the test but did not match with the theme of the game. Finally, once the activity was fully ready, a set of slides for delivering the lesson and an accompanying handout were created. This resulted in a full set of materials for an instructor to deliver the sample activity to a group of students, which was used in the evaluation as well as being part of the project output itself.

Starting from a blank Pytch project, the core of an 'endless runner' style game was implemented using some of the sprites already available in the Pytch environment. This initial prototype consisted of a car sprite on the left of the screen which could move between vertical tracks using standard keyboard input rather than the micro:bit, with Python logo sprites spawning from the right of the screen on a random track and moving left towards the car sprite. The code for this prototype was tested to ensure that it worked and was inspected to assess its complexity. The functioning prototype was relatively straightforward and was refactored to make use of variables in most places rather than constant values along with descriptive naming for functions and variables, which made the prototype easy to understand. The use of variables rather than repeated literal values also meant that it was easier to quickly tweak the prototype, which was both beneficial in the development process and for experimentation-based learning. Some additional features were added to the refined prototype, such as a scoring system, and the micro:bit was incorporated to provide the input via gestures as well as displaying the score and play sounds when the player dodges or crashes into an obstacle, resulting in a relatively complete game.

With the end result of the activity being clearly defined, of a reasonable difficulty and implemented with the aim of being understandable to the user without explanation, the lesson based around the activity was developed. Initially, the plan was for the students to implement the entire game from a minimal template, however as the plan for this was being written it was clear that this had three significant flaws. First of all, it would be very time consuming, which means it may not fit within the time window available for delivering the lesson. Second, even if it could be delivered, it would result in a lot of work before the students started to be able to properly engage with the result, which may result in them losing interest. Third, it did not encourage building understanding through exploration, but rather required students to implement code by instruction. While it may be quite satisfying to develop a working game from scratch at the end, these three flaws were significant drawbacks. Instead, it was decided to take the finished game and remove three significant components, with at least one of the components involving the micro:bit,

as fixing a broken game would still deliver a significant degree of satisfaction without the same drawbacks. Additionally, having a shorter guided portion to the lesson would allow more time for the students to experiment at the end, making the game their own and potentially achieving both a better understanding and more motivation to continue engaging with the concepts.

When deciding which components of the game to remove, there were a few considerations to keep in mind. The components removed should make a noticeable difference to the game, so that their omission is obvious and implementing them feels satisfying. However, the components should also be relatively simple to implement back into the game, so that they are not beyond the skills of the students and so that they can be accomplished within the small amount of time available. Ideally, at least one of the components should involve the micro:bit, so that the students actually engage with programming the device, rather than it just being used in the game. Removing the steering controls from the car was one obvious choice, however this would make the game completely unplayable, which is at odds with the idea of having a functioning but incomplete prototype to explore at the beginning of the lesson. Instead, the scoring, collision detection and sounds were removed, as these are all important components of the game, but do not entirely prevent it from being played. The scoring and collision detection were chosen because they are what turn the program into an actual game, with some sort of objective and a way to do better or worse. The score additionally uses the micro:bit for score display, so it is a good introduction to interacting with the micro:bit. The sounds were chosen as these improve the experience of the game and solely use the micro:bit, so introduce another major piece of the device's functionality. Additionally, the micro:bit uses a music notation style system for sounds, so it is quite easy to create custom sounds by changing notes and their durations, making it a feature particularly suited to experimentation and exploration.

With the approach to the lesson defined and the components to be implemented during the lesson selected, the actual lesson was planned. First, a basic explanation of what Pytch and the micro:bit are is provided, so that the students understand what they will be working with. The explanation should only be a short introduction, as the actual learning should be primarily hands-on. Then the students are provided with a link to the template for the game with the discussed components removed. This was done using Pytch's 'suggested demo' feature which allows for a project to be created from a hosted template. This required a build of Pytch with micro:bit support and the skeleton program to be hosted for the lesson, which the main Pytch researchers kindly facilitated. Once the students have the template open they are guided through connecting their micro:bit devices and can start the game to play around with it, being instructed to take a look through the code and see if they can understand how the game is working. Once the

students have had the chance to familiarise themselves with the game and the underlying code, they are asked to identify parts of the game that they feel are missing. Encouraging the students to identify the problem should help them engage more with the lesson and think about what they are trying to achieve, rather than just being immediately asked to implement specific features. Any missing components that are not identified can be pointed out to the students and any additional components identified can be examined later in the lesson for further exploration.

Before moving on to implementing the missing components, the basic structure of objects in Pytch is explained with reference to the code the students have examined to ensure they have the knowledge necessary to make changes. Then for each of the three missing components, the students are asked what they may need to achieve to add the component back in and are directed as to where they may need to look in the code to make the changes. Students are given a chance to write the code themselves and test their solutions to see if it works, with guidance on one method of solving the problem being provided for students to follow if they are stuck or compare with their own solutions after a short period. Explaining the thought process behind the solutions and guiding the students towards the answer rather than simply providing it is important to ensure that the learning is following a constructivist approach based on building understanding rather than a purely instructional one. Once the three components have been added, some inspiration for experimentation is provided. If the students identified additional components they wanted to add earlier in the lesson, these can then be explored now that the students have a grasp of how to modify the program. The variables controlling the different parameters of the game, such as obstacle speed and spawning delay, can be pointed out for students to experiment with, and suggestions for further components to add can be provided so that students can try implementing one of those if they are unsure of what to add. This allows the students to make the activity more of their own, and adds potential for them to continue experimenting after the lesson ends.

Up to this point, the images being used in the activity were assets that were already included in the Pytch environment, but did not necessarily fit the theme of the game, except for the car asset. A new background and an obstacle asset were required, which ended up being a three lane road and an oil barrel. A variant of the background was also created with ”Game Over” text on it, so that the background could be used to indicate the game being over rather than incorporating an additional Pytch sprite for this purpose, keeping the complexity of the project to a minimum. In addition to the two necessary assets, another two assets were also added to facilitate some of the possible additional components, such as different obstacle types and collectables, by also creating banana peel and coin assets for the students to use. All of the assets were created in Adobe Illustrator

and sized appropriately for the Pytch stage. The properly themed visuals helped to make the game feel more complete and would give the students more options to easily customise the game and experiment.

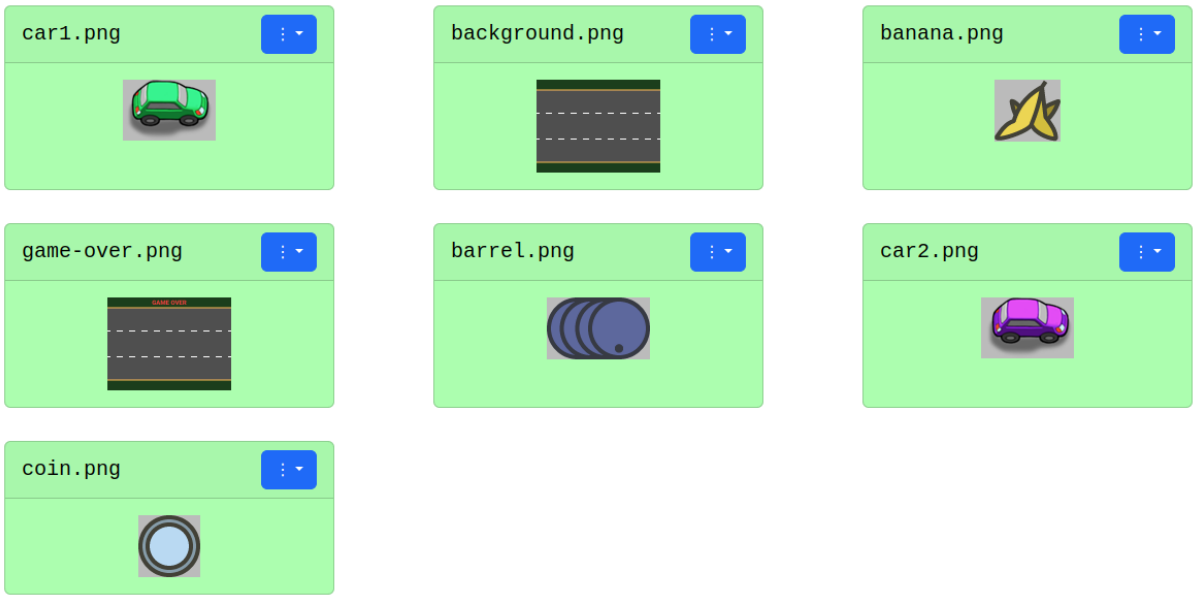

Figure 4.2: The original car sprites included with Pytch, along with the additional sprites created for the sample activity

With the activity finalised and proper assets in place, the resources for delivering the lesson were developed. Since the lesson was designed to be delivered by an instructor, the main resource for delivery was a slideshow presentation. Given that the explanations will likely need to be tailored to the skill levels of a given group, the emphasis in the presentation was on providing the core information and helpful visuals to assist with the lesson. The presentation has a visual slide for introducing Pytch and the microbit, followed by a slide with a short link to the 'Road Runner' template being used for the lesson. It then has a guide to connecting the micro:bit, with some visual aids, to help the students get set up. After these three introductory slides, the lesson should focus mainly on encouraging the students to experiment, so each slide initially only displays a prompt, such as "What's missing", with the guidance appearing after transitions to give the chance for students to engage with the prompt. This pattern is followed for the identification of the missing components and each of the three tasks to re-add components. After the tasks are completed, if the students have their own ideas and are engaging in self-driven experimentation, then the remainder of the lesson can focus on their ideas. Otherwise, there are two additional slides for encouraging and facilitating further experimentation. The first highlights some of the variables which affect the game and encourages students to try tweaking them and observing the effects. The second lists some potential 'challenges'

that the students could engage with, like the previously mentioned additional obstacles or collectables, but also including components like having multiple lives or increasing difficulty.

During the development of the slideshow presentation, it was realised that also developing a handout sheet would be beneficial for the lesson for a few reasons. First, students may want to refer back to some of the information later in the lesson, such as the steps for connecting their micro:bit if they run into an issue or the explanation of how code is structured in Pytch if they get confused. Second, the layout of the room where the lesson may be delivered was unknown, so all students having a good view of the projector screen was not guaranteed and some students may have difficulty viewing a projector screen regardless, so having a second point of reference at their desk may be important. Third, if the students want to continue working on their programs at home or in school, they may want to refer back to the link to the template, the information about Pytch from the lesson, or some of the challenges that were suggested. As a result, a handout was created using the presentation as a base, with a section corresponding to each of the slides, except for the initial introduction to Pytch and the micro:bit. Some of the information was made more concise, so that it fit neatly on the sheet, but all of the core information and code samples present in the presentation were also added to the handout sheet.

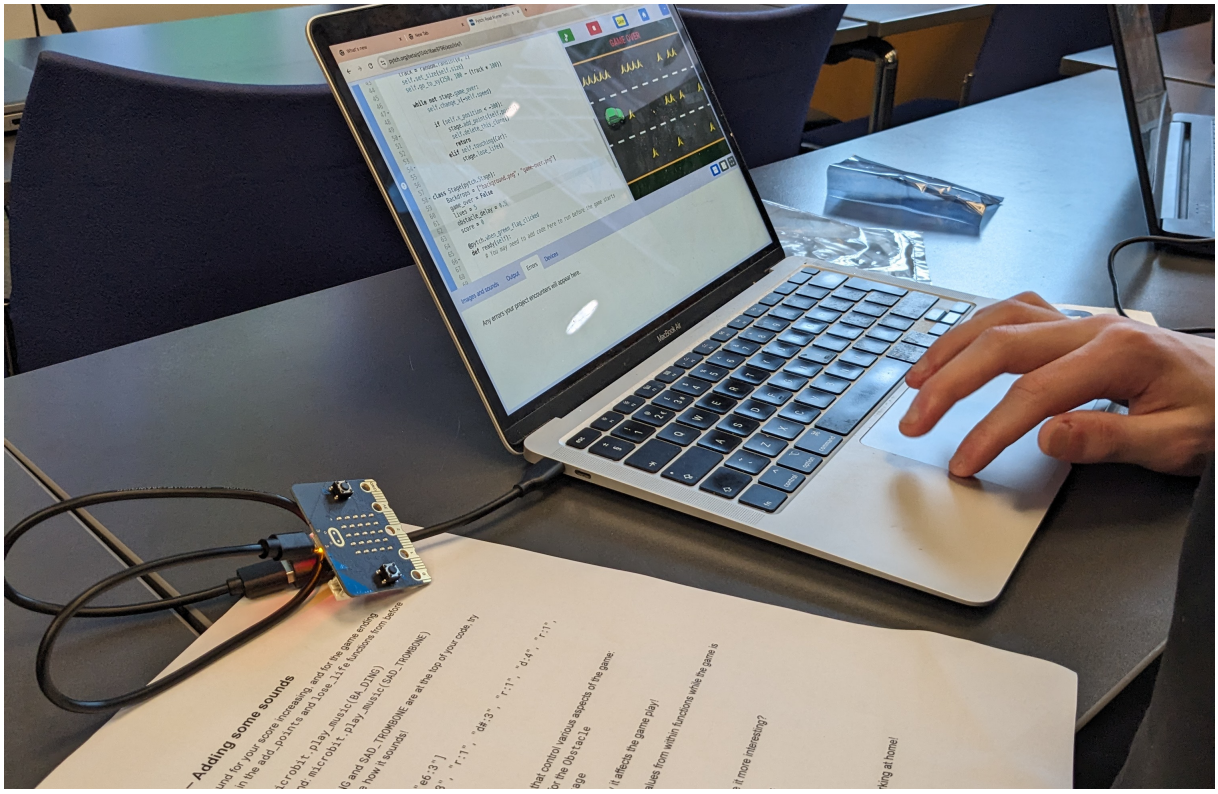

Figure 4.3: The modified Pytch environment, a micro:bit and the activity handout

4.5 Experimental Evaluation

With the sample activity fully planned, a method for experimentally evaluating the project leveraging the sample activity could be developed. This would necessitate developing a protocol for the study, as well as selecting the questions for the pre and post-activity questionnaires, preparing an observation sheet, deciding on a means of analysing the collected information, and formulating an ethics application for the study. Since the study would involve secondary school students who are minors, the study would require a full ethical review before it could be conducted, so the materials required for Trinity's research ethics approval process had to be prepared.

The protocol for the study forms naturally around the lesson plan for the sample activity. Since the main Pytch project already works with schools to conduct studies of the main environment, the same network of schools was able to be leveraged to organise the study for this project with the support of the Pytch researchers. A teacher at a school already participating in Pytch studies was contacted and a date for the study was selected. As discussed in section 3.5, a combination of quantitative and qualitative data was desired for the evaluation, so both questionnaires and an observation sheet were required. The questionnaires were created in Microsoft Forms so that the data is stored on Trinity's servers rather than on a public platform, and to simplify analysing the information from the questionnaires rather than requiring manual data entry from paper questionnaires. The observation sheets were created in Microsoft Word so that they could be printed and easily carried around by the researcher to make quick notes, then scans can be taken for reference during the analysis. Since there are a large number of ethical concerns surrounding personal information relating to minors, the study was decided to be conducted anonymously to avoid holding any personal or identifying information. In order to still correlate the responses to the pre and post-activity questionnaires, as well as observations of individual students by the researcher, the students are instead given a number for the study, which is displayed at their desk and which they enter into the questionnaire. Being able to relate the information to a single student is important to examine changes in attitudes and correlate those to the experience of that student, but doing so anonymously avoids the ethical complications of handling personal data for minors. It also enables the students to opt-out after the activity by supplying their number to the researchers via their teacher, so the information associated with that number can be removed from the evaluation.

As discussed in section 3.5, the questionnaires were based on validated instruments with extensive pre-existing research leveraging them, the Science Motivation Questionnaire II (SMQ-II), the Instructional Materials Motivation Survey (IMMS), and the Com-

puter Programming Self-Efficacy Scale (CPSES). Questions from all three instruments were selected based on their relevance to the project goals, and were sorted into either relating to motivation or learning outcomes. Ultimately, the questions from the CPSES were too general to be useful in evaluating the project, since they pertained to general ability to accomplish computer science tasks, which were unlikely to shift dramatically from a single intervention, but may be suited to a prolonged study of the efficacy of the project. The SMQ-II questions were primarily related to motivation, with some questions measuring self-efficacy potentially being both a measure of motivation and indicative of achieving learning outcomes. The IMMS questions also relate heavily to motivation, however refer specifically to the lesson completed, so may also be indicative of achieving the lesson's learning outcomes where they measure relevance of confidence. The questionnaires needed to be relatively short to have a high likelihood of being completed in their entirety, and within the time period allotted to the study, so six questions from the SMQ-II were included in both surveys to measure general motivation, and eight questions from the IMMS were included in only the post-activity questionnaire to measure attitudes towards the lesson.

The weighting towards including more questions from the IMMS was intentional, as their focus on the lesson was likely to give more concrete information about the efficacy of the activity, while changes in motivation may not be as observable after a single intervention. For the motivation questions, two questions measuring intrinsic motivation and four questions measuring self-efficacy were chosen, since these are the main areas where previous research has indicated attitude shifts from the use of physical devices [3, 9, 24]. For the lesson attitude questions, two questions measuring attention, one question measuring confidence, three questions measuring satisfaction, and two questions measuring relevance were chosen. This selection of questions focused on examining whether the specific activity was successful at engaging the students and whether it directly resulted in any further motivation for the students. The SMQ-II questions needed to be rephrased slightly to be specific to computer science rather than science generally, and in one instance the start of the question was changed to begin similarly to the others, however these changes should not have a significant impact. One of the IMMS questions was also simplified slightly to make the ending a little clearer, given the intended audience of the questionnaire. Additionally, two questions were included only in the pre-activity questionnaire to collect basic information about student gender and previous technological experience, since these were potential subjects of interest. Three open-ended questions were also included only in the post-activity questionnaire, which asked students if they encountered any issues or difficulties during the lesson, if there were any parts of the lesson they particularly enjoyed or did not enjoy, and if they had any other thoughts regarding the lesson, to provide an

opportunity to collect direct qualitative feedback from the students anonymously.

The process of completing an ethics application for a research project involved completing an application form, providing a participant sheet and consent form, which was necessary for the participant and their parent or guardian in this case as the participants were minors, writing a proposal for the research project, and including the questionnaires and observation sheet for inspection. The application form and research proposal were filled out using information from the original research proposal, goal and objectives outlined at the beginning of the project and detailed in chapters 1 and 2. For the information sheets, it was important to fully detail the study that would be conducted, but to ensure that appropriate language was used for the parents or guardians and the students respectively, as the students may require simpler language than their parents or guardians. The information sheet was based upon a set of guidelines provided by the Research Ethics Committee, detailing various aspects like the purpose of the study, who it is being performed by, how the study will proceed, how participants can withdraw, and who can be contacted with questions, based upon the protocol for the study. The information sheet for the parents and guardians was written first and then a copy was made with simplified language for the students. Since the participants are minors, an informed assent form was required for the students and an informed consent form was required for their parents or guardians. The forms were based upon their corresponding information sheets and a series of declarations based on the guidance provided by the Research Ethics Committee and the specifics of the study were composed. All of these documents were compiled into a single document for submission to the committee, and links for the questionnaires and PDF copies of the questionnaires were attached, along with the template for the observation sheet. The Research Ethics Committee responded with some follow up questions and minor recommendations for changes, and then approved the study once the questions were answered and the recommendations were adopted.

4.6 Summary

To achieve the project goal and objectives, a combination of technical, educational and research-based components were required to be implemented, all of which had to compliment each other. The discussion of each component has been ordered for clarity, but much of the work done on all components was completed in parallel throughout the duration of the project. Observations and refinements in one of the components often led to changes in the others, and the researcher continuously tested the implementation throughout the project, helping to inform implementation details and refinements. Additionally, progress meetings were regularly scheduled for most weeks over the duration of the project to

review changes with the Pytch researchers, in order to get feedback on implementation decisions and continue to improve the solution.

Chapter 5

Evaluation

This chapter will evaluate the project in two ways. Firstly, the final implementation of the project will be compared to the stated goal of the project and the research conducted to inform the design of the project to evaluate whether the project has achieved what was originally set out. Secondly, the observations and results of the two interventions conducted with the implementation, described in 3.5 and 4.5, will be analysed and used to evaluate the efficacy of the project.

5.1 Analytic Evaluation

The analytic evaluation will focus on examining the final implementation against the established goal and objectives of the project, the identified constraints, and the proposed design, with respect to the research examined in 2.1 and other similar solutions explored in 2.4.

5.1.1 Comparison to Goals, Objectives & Requirements

When designing the project, six requirements were identified in section 3.1. How well the project met each of these requirements will be evaluated in turn:

R1 (Feasibility) The solution is free to use as a contribution to an existing open source project, meaning that there is no financial barrier to a school evaluating or employing the solution. The solution leverages a device that many schools may already have a supply of, or can easily obtain in affordable class-sized kits. Additionally, the code follows a similar style to Pytch and BBC MicroPython, making it easier to learn for individuals with prior experience. Guidance for using a device is available in the Devices pane, and the API is documented in the Help pane alongside other Pytch

functionality, meaning no major instruction in the use of the solution is required outside of what is provided.

R2 (Ease of Setup) The solution was implemented as part of an existing web-based environment without the use of additional software to be installed, meaning the only step required to deploy the solution is to load the on-device firmware onto the micro:bit by dragging the provided file onto the `MICROBIT` drive available when the device is connected to a computer. The Devices pane provides troubleshooting advice to a user automatically and instructs them on how to pair a device, meaning a teacher does not need to perform the setup for each student.

R3 (Functionality) The device chosen for the solution, the BBC micro:bit, has a large number of physical capabilities, including buttons, an accelerometer, a temperature sensor, light sensor, microphone, speaker, display and GPIO. This enables a broad range of different activities which can be targeted at various skill or experience levels, from using the device as an input device to a program, performing scientific experiments using the sensors or connecting electronic circuits to a computer program.

R4 (Motivating) The solution is designed around a physical device, enabling the opportunities and benefits associated with such devices, as previously demonstrated in multiple studies [9, 10, 24]. The activity designed to accompany the technical implementation makes use of the "Use-Modify-Create" approach, based upon the learning theories of constructivism and bricolage and their benefits [11, 14].

R5 (Verifiable) The interventions conducted indicated the solution was effective in motivating students and the students were able to successfully complete the activity. Although the results are only indicative due to the time constraints on the project, the initial results are promising and the teachers of the classes involved were interested in implementing the solution in their own lessons and curricula.

R6 (Idiomatic) The design and implementation were carefully planned to ensure that the output of the project would abide by the design principles and conventions of Pytch and facilitate transitions between Pytch and the native programming interface for the device.

Moving to the objectives, the implementation was able to meet all of the stated objectives in section 1.3. The potential physical computing devices were thoroughly researched and evaluated, and software exposing all of the relevant functionality was developed for the chosen device, enabling communication with Pytch. The interconnect functionality

was implemented into the Pytch environment, and the interface developed to manage the connection with the device was simple and easy to use for an inexperienced user, requiring minimal instruction during the interventions conducted. The library and runtime support implemented to allow users to interact with the device was effective and intuitive to use for the students the evaluation was conducted with. The sample activity developed was effective at engaging the students and showcasing some of the developed functionality within the constraints on the interventions and the project more broadly. Analytic and experimental evaluations were conducted for the project, indicating success at achieving the other objectives despite the time constraints and the potential merit of this approach and its further study. As a result of fulfilling all of the objectives of the project, the overall goal of the project described in section 1.3 is considered to be successfully achieved, as support for a readily available computing device with a broad set of features, the BBC micro:bit, has been implemented into a programming environment oriented towards inexperienced users and educational use, Pytch, which was then evaluated with respect to student motivation and learning outcomes for the stated audience of secondary school computer science students.

5.1.2 Comparison with Alternatives

Comparing the implemented solution with the alternative solutions identified in section 2.4, there are some clear differences and distinct improvements which create a unique value for the project within the computer science education space. Compared to the support for physical devices in Scratch, the functionality available on the device is more fully exposed than in Scratch. Scratch only supports the functionality available on the original revisions of the micro:bit, which does not include additions like the microphone, speaker or touch-sensitive logo. Additionally, Scratch does not provide support for multiple sensors that were available on the original revisions, such as the temperature sensor or light sensing mode of the display, does not support the radio, and only provides limited support for the GPIO. By contrast, the project supports all of these capabilities, in similar manners to the MicroPython API, enabling a more comprehensive range of activities similar to programming the device directly, but still enabling the device to be easily integrated into multimedia projects through the Pytch environment. Compared to the editors provided for programming the micro:bit directly, the official micro:bit Python Editor and Microsoft MakeCode for micro:bit, the project provides a similar level of functionality, with the exception of some low level communication protocols, but incorporates the device into a broader environment rather than programming it directly. This approach has multiple benefits compared to direct programming, including faster iteration, easier integration

into existing projects, and new possibilities not feasible solely programming the device itself. While there is also distinct value to programming the device directly, this hybrid approach retains most of the benefits for an inexperienced programmer while familiarising them with the concepts necessary for direct programming. Finally, compared to previous attempts to make hardware programming more accessible like Modkit, the project retains the broader electronics capabilities and ease-of-use of these solutions, while still using a textual programming language rather than a block-based one, facilitating a user's transition towards textual programming and traditional direct programming in the future, while also leveraging the benefits of integration into a broader environment as previously mentioned. These differences make the project distinct from alternatives, providing unique value and approaches for leveraging physical devices in computer science education.

5.2 Experimental Evaluation

Two interventions were conducted as part of evaluating the project. The first intervention was an unexpected opportunity that arose during the project to run a workshop as part of a larger event being run by the School of Statistics and Computer Science (SCSS) in Trinity College Dublin. Data collection was not possible at this workshop due to time constraints, but it provided valuable insights into the project before it had been fully implemented. The second intervention was a planned session using the final implementation, coordinated in advance with a teacher already participating in studies with the Pytch project, and involved quantitative and qualitative data collection.

5.2.1 Outreach Workshop

As part of an outreach day run by the SCSS with a 5th year Leaving Certificate Computer Science class from a fee-paying school located in Kilkenny, a workshop was conducted using the technology and activity developed for the project, as described in chapter 4. The workshop lasted about 45 minutes. The students had some limited experience of programming in Python, but did not have experience with Pytch or the micro:bit. Since the opportunity was presented without sufficient time to distribute and collect consent forms for the participants and because the time allotted was relatively short, formal data collection was not possible for this workshop, but the workshop followed the same lesson plan and made use of the same materials as planned for the classroom study. Observations were made during the activity based on the engagement, reactions and feedback from the students, which provided some valuable insights into the efficacy of the project, which were then used to refine the implementation and activity before the classroom study.

Students brought their own laptops that they would typically use in school, which were primarily Windows devices, except for one student with a MacBook. A micro:bit device pre-prepared with the on-device software was provided for each student in order to maximise the amount of time available for completing the workshop. Guest Wi-Fi credentials were provided by the college, however some students had difficulties connecting to the internet using their laptops and had to work with a student beside them instead. This only affected three students out of the twelve present, and these students were still able to engage with the workshop by collaborating with another student. Given the short amount of time allowed for the workshop and the lack of control over the students' devices, there was not an alternative solution available to enable the students to fully participate.

The students responded well to the workshop and were able to successfully complete the entire activity with only minor issues. The main issues encountered could be split into difficulties writing the Python code and problems encountered with the technical implementation. In the first category, most of the difficulties were due to the students' limited experience with Python, meaning that they made minor syntax or functional errors. These issues were largely alleviated by the code snippets provided in the presentation and the hand out sheet or through student questions, although a few students had trouble distinguishing what was part of the code snippets and what was the surrounding text. In the materials, a monospace font had been used for code snippets and a normal sans-serif font used for all other text, but this was not an obvious enough distinction for some of the students. One refinement made to the activity materials was to change the colour of the code snippets for the presentation and separate them more clearly in the hand out sheet. In the second category, some students encountered connectivity issues with their micro:bits, which were mostly solved by disconnecting and reconnecting the devices. Some students did not see the error messages that would help them diagnose the problem due to an issue with the reactivity of the device state and required assistance. Some students also encountered issues when dramatically increasing the speed of their game, as the messages being sent to the micro:bit began to collide, resulting in their game crashing. Both of these issues were solved after the workshop by making adjustments to the code. In the first case, the way the device state was updated was modified to ensure it always triggered correctly, which was then tested using fuzzing to ensure it was working. In the second case, there had been a queuing system for handling command responses but not for sending commands, meaning two commands could theoretically be sent at the same time. This was resolved by introducing queuing for outgoing commands as well, ensuring that there could not be any conflicts. A related issue found while debugging the collisions was that some of the commands would block execution for longer than a student might expect and cause the device to appear unresponsive, in particular the music command

which was used heavily in the activity, so this was also fixed by changing the parameters in the on-device software.

Figure 5.1: Students using the implementation in the initial workshop

The students visibly enjoyed working on the game, displaying satisfaction when accomplishing each of the tasks and going beyond the tasks to customise the game, making it their own. As was mentioned in section 4.4, additional assets were created to enable further experimentation, however students also made use of the assets built into Pytch, such as one student who made the obstacles show up as birds rather than the default oil barrel. Many students engaged in customisation of the gameplay, changing the variables controlling the sizes and speed of objects and the rate at which obstacles spawn. Students appeared to enjoy being able to make their game different from their classmates', so one refinement made for the classroom study was to incorporate a wider range of assets for students to use, including an apple core, a gem and additional colours for the barrel and coin. The use of the micro:bit also appeared to get the students more engaged with the activity, as they made larger and more dramatic gestures than were actually necessary to control the game with the micro:bit, suggesting a deeper engagement with the game arising from the physicality, which would be unlikely to occur with the use of keyboard controls. Since there was some additional time at the end of the session and the visiting school wanted to award a prize, the students were asked how they might implement one of the challenges mentioned in section 4.4, adding multiple lives to the game. One of

the students was able to provide a comprehensive answer based on their examination of the code despite the group's unfamiliarity with Pytch and win the prize, indicating that the students were also able to grasp the environment relatively quickly. Some students indicated an interest in working with Pytch and the project further at home, and the teacher expressed interest in using the activity and the functionality developed in their classes, so they were given additional copies of the handout sheet to be able to replicate the workshop themselves and explore the modified version of Pytch. This interest in engaging with the project further is a particularly positive indicator of the project's efficacy in motivating students.

5.2.2 Classroom Study

A study was organised with a Transition Year class in a fee-paying school located in Dublin, which had previously participated in studies of the Pytch environment and which used Pytch in some of its classes. In addition to the workshop delivered as part of the previous intervention, this study also involved the use of the questionnaires designed and developed in sections 3.5 and 4.5, respectively. The students represented a mix of experience and interest levels and have not yet selected their subjects for the Leaving Certificate, meaning that they have not necessarily settled in their views on computer science. This session took place over about 50 minutes, slightly longer than the previous workshop, however the setup took additional time due to the larger class size, and the questionnaires also took about 5 minutes each to complete, so the time available for the activity was closer to 35 minutes. This is a significant reduction from the previous 45 minutes, especially given the larger class size. This reduced the time available for explaining the code to the students and for encouraging experimentation at the end, which may have reduced the benefits of the activity, as is reflected in some responses to the questionnaires.

Of the 20 students who participated in the session, 11 students completed both the pre and post-activity questionnaires, with one student completing just the pre-activity questionnaire and one student completing just the post-activity questionnaire. For the purposes of this evaluation, we will consider mostly the 11 students who completed both questionnaires. Of these students, 7 were female and 4 were male. All students reported having some experience of computer science, with 7 students reporting having used a micro:bit before and 9 students reporting having either taken a programming class or written a program before. The students were grouped into either having 'limited experience', meaning they only engaged in 1 or 2 of the listed computer science activities, or having 'substantial experience', meaning they engaged with 3 or more of the activities.

There were 8 students with limited experience, 5 female and 3 male, and 3 students with substantial experience, 2 female and 1 male, providing a similar distribution of female and male students in both experience groups. All of these students except for 1 reported having successfully completed the activity, with this student not providing a response to the question. Based on observations made during the session, all students made significant progress through the first two tasks, adding a score system and a game over feature to the game, and most students were able to progress through the third task, adding music to the game, though more students struggled with this task. Due to the small sample size of respondents and the results being from a single intervention, all of these results are purely indicative, but they do provide some patterns and useful insights regarding the approach taken in the project and the efficacy of the implementation.

Beginning with the motivation questions, the same 6 questions assessing student motivation to study computer science were asked before and after the activity, with 2 questions being asked to assess intrinsic motivation to study computer science, and 4 questions being asked to assess self efficacy. Of the 11 students who completed both questionnaires, 6 students showed an improvement in motivation, with 3 students showing no change in motivation, and 2 students showing decreases in motivation. Of the 3 students showing no change, 1 of the students was in the substantial experience group and had reported the highest levels of motivation in the class in the pre-activity questionnaire, and 2 of the students were in the limited experience group and had reported two of the lowest levels of motivation in the pre-activity questionnaire. Of the 2 students who showed a decrease in motivation, one student did not complete the full set of motivational questions in the post-activity questionnaire, making the actual change unclear. The other student showed a slight decrease in self efficacy and responded neutrally about the lesson in the lesson-focused questions. The vast majority of the improvements in motivation came from increases to self-efficacy with 7 students reporting an improvement in self-efficacy, 3 of whom improved in 2 or more of the questions. The main self-efficacy questions which improved were whether the students believed they could understand computer science and whether the students were confident in their ability to write a computer program. Of these 7 students, 5 of them were in the limited experience category, a majority of this category, and indicated low initial self-efficacy, which could suggest that the activity was helpful at making computer programming seem more feasible to students with limited confidence. The 2 students with substantial experience already had high self-efficacy scores for the class group, but these increased further. Only 4 students showed increased intrinsic motivation to study computer science, with all increases occurring in the question asking whether students enjoyed computer science, 3 of which were students with substantial experience. The single student in the limited experience group was one of

the two students in this experience group to initially indicate a high interest in studying computer science. This suggests that the activity may have helped to cement pre-existing interest by making an activity students were already interested in seem more enjoyable. Of the 7 female students, 5 showed an improvement in self-efficacy, with one student showing a slight improvement in her confidence writing a computer program and a slight disimprovement in her belief she could master computer science, and one student showing only a slight disimprovement in her belief she could master computer science, who was the student with the highest self-efficacy in the class group before and after the activity. This would suggest that the activity may have been effective at improving female students' self-efficacy.

Moving to the questions measuring attitudes towards the lesson, students generally responded positively towards the lesson on the metrics of attention, confidence and satisfaction, but responses were more mixed on the metric of relevance. 6 students agreed or strongly agreed with the statement that the lesson had stimulated their curiosity, with the remaining 5 students remaining neutral on the statement. 7 students disagreed or strongly disagreed with the statement that the amount of repetition in the activity caused them to be bored sometimes, with 2 students remaining neutral and 2 agreeing. One of the students who agreed indicated the lesson did stimulate his curiosity and that he felt a satisfying feeling of achievement, as well as displaying improvements in self-efficacy and intrinsic motivation to study computer science, while the other student was more neutral across the board and was one of the two students to disimprove in self-efficacy. 9 students disagreed with the statement that the lesson was more difficult than they would have liked, with 2 students remaining neutral. These results would generally suggest that the activity was able to hold the attention of the students and that the difficulty level was appropriate to the student group participating in the lesson, and that the results could likely be improved upon by focusing further on the customisation of the game in a longer session. 6 students agreed or strongly agreed with the statement that completing the lesson gave them a satisfying feeling of achievement, with 2 students remaining neutral and 3 students disagreeing or strongly disagreeing with the statement. The 3 students who disagreed and one of the students who remained neutral all also remained neutral on the statement that the lesson had stimulated their curiosity, which represents a possible correlation and could suggest a lack of interest in the activity as a cause, which could mean that other activities would be more effective. 2 students strongly agreed or agreed with the statement that they really enjoyed the lesson, 8 students remained neutral, and 1 student disagreed with the statement. Of the 2 students who agreed, both of them were in the substantial experience group, suggesting that they potentially got more out of the activity as a result of their prior experience, and that more detailed explanation and a

longer session may be beneficial for other students to benefit similarly. Of the 8 students who remained neutral, 6 indicated that they enjoyed the activity in the open comment questions, and 4 students who expressed enjoyment also indicated that they struggled with the activity, potentially suggesting that the difficulties faced balanced out some of the enjoyment they felt. 4 of the 8 students who remained neutral did indicate that they found the activity satisfying, including 3 of the 6 students who expressed enjoyment in the comments. This would further suggest that a slower pace with more explanation and opportunities to experiment with or play the game may have helped their enjoyment of the activity.

The responses to the question about whether students enjoyed the lesson enough so much they would like to know more were more mixed, with only 2 students agreeing with the statement, 3 students remaining neutral, and 6 students disagreeing or strongly disagreeing with the statement. Of the 9 students who were neutral or disagreed, 4 indicated they found the lesson satisfying and 4 indicated the lesson stimulated their curiosity, which may suggest the different possibilities available with the micro:bit and further work on the game was not clear enough to interest the students in further study, even if they were positively inclined towards the individual activity. Replacing one or two of the tasks with more novel uses of the micro:bit or experimentation with the game may have helped improve this. Relevance was one area where the responses were quite mixed. Only 1 student agreed with the statement that the lesson was not relevant to her needs because they already knew most of it, with this student being in the substantial experience group, and 2 students were neutral on the statement. One of the students agreed that the lesson stimulated her curiosity and that it gave her a satisfying sense of achievement, with the other student disagreeing that he enjoyed the lesson and that he felt a satisfying sense of achievement from completing it. This would suggest that the overlap with prior knowledge is not necessarily detrimental to the students' experience, though a strong overlap is likely to be detrimental. The remaining 8 students disagreed or strongly disagreed with the statement, indicating that this was a novel activity for many of the students, which aligns with the generally positive responses towards the lesson stimulating their curiosity. Only 4 students out of 11 strongly agreed or agreed with the statement that the material was relevant to their interests, with 3 of the students being in the substantial experience group and having indicated that they were interested in studying computer science before in the pre-activity questionnaire. The remaining 7 students all disagreed or strongly disagreed with the statement and were all in the limited experience group, with only 2 of these students indicating interest in studying computer science in the pre-activity questionnaire. 4 of these students responded positively to the activity, with 3 students indicating that the lesson had stimulated their curiosity and 3 students indicating

that completing the lesson had given them a satisfying feeling of accomplishment. These 4 students also all showed increases in self-efficacy after the lesson. This would suggest that, so long as the activity is suitably interesting or satisfying, the students do not need to have been initially interested in the area to benefit from the activity, and these students may have experienced an increased interest in the area if some of the previously identified deficits with the activity were addressed or they participated in more sessions.

The students also had three opportunities to provide comments In the post-activity questionnaire, whether they encountered any issues or difficulties, whether there were any parts of the exercise they particularly enjoyed or did not enjoy, and whether they had any other thoughts regarding the exercise. The main difficulties cited were to do with the programming, with 6 students mentioning difficulties they had with the code in some way. 5 of these students had agreed or strongly agreed that the lesson gave them a satisfying sense of achievement, and all 6 students disagreed that the lesson was more difficult than they would have liked, indicating that the difficulty was present but manageable, which is ideal to ensure the students develop their skills without being discouraged. 5 of these students experienced increases in self-efficacy after the activity, with one student who initially had very low self-efficacy scores for the class group improving in three of the four self-efficacy questions. The final student experienced a slight decrease in self-efficacy, but had the highest initial self-efficacy score before and after the activity, and did experience a slight increase in intrinsic motivation to study computer science under the question asking whether students believe they would enjoy studying computer science. 3 students also mentioned difficulties with the micro:bit, 2 of which were regarding using it to steer in the game and one regarding difficulties connecting the micro:bit. Although the setup process for the micro:bit was explained as part of the lesson, the projector screen was not easily visible to all students and the student could have easily missed this step given the additional time pressure. One improvement to help mitigate this could be to provide a more guided setup experience from within Pytch that provides more explanation, which was initially not implemented due to concerns about overcomplicating the interface. The 2 comments about the steering could be related to the fact that the micro:bit's gesture events were being used so the device needed to be turned on its side to register a gesture, which some students may have found cumbersome. While it is possible to use the accelerometer directly to control the steering, it would have further complicated the code, so would be more suitable for exploring in a follow-on activity as a further improvement to the game. Both of these students still agreed or strongly agreed that the lesson gave them a satisfying sense of achievement, and one student strongly agreed that she enjoyed the lesson. One student commented that he felt the instructions should be clearer, but disagreed that the lesson was more difficult than he would have liked, indicating that there may have been

frustrating elements that were not necessarily difficult, which may need to be addressed. Given most students were able to follow the instructions effectively and complete the activity, it is likely that the right level of instruction was provided for the class overall and more time would have allowed for more tailored assistance for students having difficulties. No students reported or were observed encountering issues with device command collisions or the micro:bit becoming unresponsive, indicating that the improvements made to the interconnect after the previous session were effective at solving the problems.

The responses to the question about what students enjoyed or did not enjoy were primarily positive, with all 7 responses focusing primarily on aspects of the activity they enjoyed. Of the 7 students, 2 students specifically mentioned having fun playing the finished game, 2 students specifically mentioned enjoying fixing the game, and one student mentioned the activity reinforcing her wanting to make her own game. This suggests that the game was an effective activity at engaging the target audience since students responded positively to this aspect. One student who reported having difficulties during the activity mentioned that she "liked trying to figure out the code [herself] despite [her] lack of knowledge in computer science and coding", and did indicate an improvement in self-efficacy. One student indicated that they specifically "enjoyed changing the sprites and variables" in the game, and another student indicated that they specifically enjoyed "the fact we got to code into the original code and improve it". These answers, alongside the answers highlighting specifically fixing or improving the game, suggests the bricolage approach was effective at engaging the target audience. One student who was neutral on his enjoyment of the lesson and interest in knowing more commented that "if the game was more interested *[sic]* it would be more interesting". This reinforces the idea that building further on the game is something students who were less motivated by the activity might engage with more, and reinforces the previous observation that more experimentation may have helped to increase students' enjoyment of the lesson.

During the lesson, a few additional observations were made. One student was observed engaging in customisation early in the lesson, working ahead of the class using the handouts provided and the help pane integrated into Pytch, which was pointed out to the students at the beginning of this lesson. This student indicated a strong initial interest in studying computer science, reported the lesson was relevant to her interests and also reported the highest satisfaction with and enjoyment of the lesson in the class, so her self-directed learning is unsurprising. The student's customisation of the game drew the attention of one of her neighbours, who she then showed how to make the modifications. The neighbour then proceeded to make similar visual and gameplay customisations, and was the student who commented that they especially enjoyed changing the sprites and variables. The neighbour also reported high satisfaction with the lesson, and indicated

increased interest in studying computer science after the activity. Collaboration was common during the lesson, with students looking at what their neighbours were doing or asking them for help. This was facilitated by both the visual nature of the game on the Pytch stage and the use of the micro:bit. Two students with limited experience and low self-efficacy before the activity who worked together throughout the lesson both indicated improvements in self-efficacy after the lesson. Integrating collaboration more directly into the lesson could potentially amplify these effects more broadly. Although not used by all students, the handouts were effective at allowing the more experienced students to work ahead and experiment, and at helping guide the students who were experiencing difficulties with the activity, making them a worthwhile component of the lesson. There were also multiple celebratory moments observed, where many students exclaimed about or were excited by their progress, which is likely reflected in the positive response from many students about their satisfaction from the lesson. After the activity, the teacher indicated her own interest in exploring the solution further and incorporating it into her lessons and curricula.

The evaluation indicates a primarily positive response from most of the students participating in the session, across gender and different levels of experience. Only 3 students of the 11 who completed the post-activity questionnaire indicated that they did not either enjoy or feel satisfied after completing the activity, one of whom had extensive experience and had indicated he was interested in studying computer science in the pre-activity questionnaire, for whom the activity may have been too straightforward. The shortcomings identified through the evaluation and the comments submitted by the students point to issues with the delivery, scope and complexity of the lesson, rather than issues with the physical component of the activity itself. This is reinforced by the observations made during the lesson and the generally positive response to the lesson from most students, as well as improvements in self-efficacy, especially among female students, even if the response was not unanimously positive. It is clear that the ways in which the capabilities developed are used as part of workshops, lessons and a broader curriculum will need to be examined and studied further, which will be discussed in section 6.2.

5.3 Summary

While the small sample sizes and durations of the two interventions mean their results are solely indicative, they do suggest that the approach can be effective at capturing the interests of and engaging students, especially female students. Having achieved the goal and objectives of the project, the implementation is now ready to be used and examined further. Based upon the evaluation, areas of refinement and further development for the

project and its evaluation have been identified, which will be explored in chapter 6.

Chapter 6

Conclusions & Further Work

This chapter will present the final conclusions reached as a result of the design and implementation of the project, and propose potential further work to be carried out in order to improve upon and better study the project.

6.1 Conclusions

The achievement of the project's goal has broadened the possibilities of the Pytch environment and developed a fully-featured hybrid programming environment for beginners, incorporating physical devices and Scratch-like multimedia capabilities, one of the first of its kind. Similar existing approaches like using the micro:bit in Scratch via Scratch extensions are far more limited in their capabilities, not fully exposing the capabilities of the device, and do not appear to be actively worked on. The implementation of the project is complete and functions well, having been tested extensively throughout the project by the researcher and by the research team working on the Pytch project, and has been successfully tested twice with the target users. The design process, refinements made during implementation and the evaluation of the output may help to inform the development and evaluation of other projects in the areas of computer science education. The developed lesson and accompanying resources also form an evaluated base to develop further lessons on top of, allowing for the development of a full hybrid programming course and informing potential related or alternative approaches to integrating physical devices into computer science education.

The two interventions conducted with the project are indicative of a promising approach to integrating physical devices into computer science education in a simple but comprehensive manner. In the first intervention, the activity was able to engage a class of students already studying computer science in secondary education, having visible results with students being excited about getting the game to function and playing their creations,

which they customised and adapted to their own preferences. Although quantitative data collection was not possible, feedback from the teacher and students indicated that they would be willing to engage with the system again through further activities. In the second intervention, the activity was able to engage a group of students who were considering studying computer science in secondary education the following year, with a variety of different experience levels. The quantitative data collected indicates that the approach was effective at stimulating the students curiosity and giving them a sense of satisfying accomplishment, as well as improving their sense of self-efficacy, which is essential for students' pursuit of computer science, especially where they may not be as encouraged or supported, such as in the case of female students. The promising engagement with the activity but limited indications of improved motivation to study computer science or pursue further projects suggest that further development of the activities and multiple interventions are necessary to make the impact more substantial for a broader range of students, but that the potential to do so is there. The benefits for female students in particular align with previous studies in the field, and given the continued underrepresentation of women in computer science education and professions, indicate merit to further study.

The evaluations of the project were limited in time and in scope as a result of broader time constraints on the project and the logistical difficulties associated with organising sessions with students, requiring coordination with schools and acquiring consent from parents or guardians. A single session is not a sufficient amount of engagement to measure longer term and deeper changes in motivation, interest and self-efficacy, and is unlikely to be as effective at obtaining the desired benefits as a longer course or programme. The sample sizes available were also limited, making it difficult to evaluate the overall impact of the interventions, as well as the impact based on different characteristics, such as gender or prior experience. Both schools engaged in the interventions were also fee-paying schools, which are not necessarily representative of the wider student and teacher population, who may have different or additional needs or considerations. Additionally, only one type of lesson was able to be evaluated, which may have appealed mainly to students with a specific background, rather than testing a range of different lesson types and approaches.

Ultimately, the approach used seems promising, in line with what was hypothesised at the outset of the project, based on the existing research in the area regarding the use of physical devices for computer science education. Previous studies into physical devices have indicated that it is an effective means of engaging students, especially inexperienced and female students, who are less catered to by traditional approaches to computer science education [9, 10, 24]. The preliminary evaluations reinforce this hypothesis, as many

students involved in the interventions had limited experience with computer science and the female students in the second intervention appear to have particularly benefited from the approach, although these results require further study to be considered more than potentially indicative. This approach was able to overcome some of the logistical challenges identified in previous research, by utilising the device as a peripheral to an environment on a computer and using web technologies to implement the support, avoiding the need to install or configure software on a user's computer or engage in slow iteration cycles programming a device [5, 12, 19]. Given the ease of setup during the sessions despite students using their own or the school's devices and the lack of technical issues encountered with the device during the activities, it seems that the project was effective at overcoming these deficits of previous approaches. While mandating that the device remains attached to the computer does introduce some limitations of its own, it provides an intermediate step before direct programming that makes it easier for an experienced user to begin using a physical device and learn the core concepts.

In line with previous educational research and course studies, the experimental bricolage approach taken by the activity developed as part of the project was effective at getting students immediately interested in the activity, avoiding disengagement from long periods of coding before seeing any results and encouraging students to engage in self-driven experimentation [7, 11, 26]. The students were able to effectively examine the existing code and understand how it worked in order to make the necessary modifications, as well as make modifications of their own. The process of achieving visible and tangible results at each stage of the lesson ensured most students stayed engaged and got a sense of satisfaction as they were able to interact with their work. The sense of accomplishment was amplified for students who made the game their own through their customisations, changing the aesthetics, rules and control of the game. This supports the hypothesis that this approach can be an effective means of teaching introductory computer science classes, and that the use of a physical device can be an effective means of encouraging this approach by acting as a bridge from an existing program into the physical world.

6.2 Further Work

While the project has been ongoing, the Pytch project has developed and released the second version of the Pytch environment, incorporating a new coding method which allows a user to program 'script-by-script' rather than through the class and decorator system used in the initial version of the environment. Since the project began before this work was in a finalised state it was based upon the original version of the environment. Now that the new version has been released as of the completion of the project, the implementation

needs to be rebased on top of the updated environment in order for it to be merged back into the main Pytch project, which is the most important piece of additional work. Additionally, the on-device software for the micro:bit is currently built independently and flashed onto the devices using a script, rather than being incorporated into the Pytch build process and distributed via the environment. Work needs to be coordinated with the Pytch team to integrate the build process developed for the on-device software into the overall build process for the environment so that the software can be made easily available in some way, prior to the general release of the functionality.

As identified in section 6.1, there were a number of limitations with the evaluations of the project conducted, which could be overcome through subsequent, more comprehensive evaluations. The primary limitations identified were the small sample sizes of the interventions conducted and the inability to verify whether benefits continued beyond a single intervention. Both of these limitations could be overcome to a substantial degree through running courses using the implementation of the project, such as single week summer programmes, as used in some previous studies [3]. Conducting multiple more in-depth interventions would result in more robust results and would also allow for a better study of the different factors, which may influence the efficacy of the approach and the groups of students for whom the approach is more or less effective. The specific designs of the lessons or the subject areas they focus on within computer science could also be altered to examine how the benefits may differ in different cases. Another method of evaluating the extended impact of the approach would be to run a programme taking place through regular sessions over an entire term, either as independently or together with a school. Such an intervention would require a substantial design phase involving consultation with teachers in order to develop a suitable curriculum, but would enable a representative evaluation of how the approach could be employed in a classroom setting and whether the benefits are maintained over a prolonged period of time.

Independent of additional evaluations, the development of additional activities and accompanying resources in a similar manner to the activity already developed would facilitate the use of the project by teachers and instructors in their own classes and programmes. While opportunities for refinement of the implementation have been identified and will be discussed in the following paragraphs, the evaluations indicate that the implementation is in a usable and beneficial state already, which would benefit from additional supplementary resources. The developed activity focuses on using the micro:bit to extend a simple game, but other activities involving additional components of the micro:bit, such as electronics activities leveraging the GPIO, networked activities leveraging the radio or scientific activities leveraging the onboard sensors would all help to showcase the potential of the project. The Pytch environment features a tutorial system which allows

independent students to learn programming and the specifics of the environment directly through an integrated system, and has been substantially revised in the new version of the environment, enabling more detailed and sophisticated tutorials. Incorporating the activity developed for the project into this new tutorial system, as well as any further activities developed in the future, would make it possible for independent students using Pytch to learn how to use the micro:bit capabilities as well.

While the implementation provides a relatively complete programming interface to the micro:bit and user interface for managing connected devices, there are substantial additions that could be made in order to further improve the ability for students to work with the micro:bit. Some form of simulator for the micro:bit, similar to what is available in the official micro:bit Python Editor and Microsoft MakeCode for micro:bit, would enable students, teachers and instructors to explore the micro:bit functionality in Pytch without having access to a device, which may help them to evaluate whether the capabilities would be useful or interesting to them before purchasing a device, potentially helping encourage adoption. Inclusion of such a simulator could also facilitate end-to-end testing of the micro:bit functionality, assuming it provided an accurate simulation of the device, as it would still test the majority of the functionality without requiring a physical device to be connected, making testing simpler and more consistent. An inspector, which enables the user to see the underlying messages being sent to and received from the micro:bit, could help users diagnose problems with their programs, as well as help developers debug issues with their device or the environment. This could be implemented as a stream of events presented in a user-friendly way, such as indicating whether commands have finished executing and any response received. As identified during the second intervention, the inclusion of a more guided setup experience for a user with no micro:bit connected may also help students unfamiliar with Pytch and independent users set up their devices more easily. Additionally, although connecting via USB makes sense in the majority of cases, providing a wireless mode could also remove some of the physical limitations imposed by using a tethered approach, enabling almost all uses possible with direct programming. Facilitating this as an additional option rather than the default still avoids the issues identified with wireless connectivity in section 3.3 while providing the flexibility to allow its use when required.

As well as adding additional functionality for the currently supported device, adding support for other popular devices could be beneficial to facilitate wider adoption and enable more projects, as different devices may have different capabilities and users who already have certain devices may not want to purchase a new device just to use this functionality in Pytch. The Adafruit Circuit Playground has similar functionality available to the micro:bit, with some differences that may make it more suited to electronics projects,

so would be one possible candidate for support. Arduino boards are also very popular and could enable a large number of projects through their extensibility and pre-existing accessory ecosystem, so would also be a strong candidate for support in order to expand the possibilities available in Pytch. Adafruit supports the Circuit Playground via their CircuitPython variant of MicroPython, among many other devices, and multiple Arduino boards are also supported through either MicroPython or CircuitPython, meaning a common base could be developed from the micro:bit implementation and only device-specific functionality would need to be reimplemented. Alternatively, moving to C-based on-device software would allow for supporting many different devices with a common base, as most devices support direct programming in C, and this could enable additional possibilities in terms of performance and functionality. Implementing the initial on-device software in C would have taken substantially more time which would not have made sense before the approach was able to be evaluated, but might make sense when expanding support to multiple devices since there are indications this approach is effective.

Alongside identifying further work building upon the project directly, additional functionality for the Pytch environment which would support the approach explored in the project and the use of external devices has been identified through the evaluation of the project. Some form of debugger or inspector for projects would be beneficial for helping students to debug problems with their code, as well as having the benefit of teaching them valuable skills that transfer to more advanced computer science projects and courses. A simple interface could be provided, allowing a user to inspect the state of the program while they step through it instruction by instruction, while hiding or simplifying some of the complexity like the call stack and non-user variables. Prompting the user to re-run with the debugger enabled when their project encounters an error, automatically inserting a suitable breakpoint, and providing a simple interface tour to this functionality would facilitate introducing the techniques of debugging in an approachable manner for an inexperienced user. Simplified debugging in this manner naturally fits in with a bricolage approach, as it enables a user to explore the code and see what exactly is happening, as well as potentially making experimentation less frustrating by providing a straightforward method to debug problems. Additionally, inexperienced programmers often struggle to understand how to use a debugger effectively, so a simplified introduction to how debugging works may help users familiarise themselves with these concepts, as well as motivate them to explore the use of debuggers in later projects.

A smaller addition identified that could be beneficial would be a notification or alert system, that would allow the environment to notify the user of potential problems. Currently, a program encounters an error if it uses micro:bit functionality without a device correctly connected and the prompt to resolve this only appears in the device pane, which

may not be spotted initially by a user. Providing some way to indicate that a device is not connected or has an issue before the user encounters an error would allow them to preemptively fix the issue and avoid the frustration associated with encountering an error when using their program. While encountering a single error is not always a major source of frustration, encountering errors repeatedly when a user is trying to get their program to work can lead to substantial frustration and ultimately attrition, so preventing avoidable frustration from occurring in the first place is desirable. Additionally, if an error only occurs late in the execution of a program it requires the user to re-run the program up to that point in order to test their changes, which can amplify frustration if this time is substantial, since the user has to spend significant time waiting to see if they will encounter an error. Such functionality could also likely be used in other areas, such as for linting of a user's program to identify common issues, which would also help to alleviate this type of frustration by providing live feedback to the user.

6.3 Final Remarks

The output of the project is a successful implementation of a hybrid programming environment incorporating both physical devices and multimedia capabilities. This is a different approach to what is primarily being studied in the field currently, which focuses more on the benefits of directly programming hardware and making this more accessible to inexperienced users. The approach explored in the project has its own benefits and the evaluation of its implementation suggests that the approach can also be effective in similar ways to existing alternatives, but can expand the possibilities feasible to explore in a classroom with inexperienced students by simplifying some logistical aspects and the complexity of using physical devices. There is a range of possible further work building upon the project, spanning additional evaluation, further technological development, creating additional learning materials, and improving the underlying environment the project builds on top of in complementary ways. Many of these would present substantial improvements to research or tools available in computer science education and would make the benefits of the approach used in the project clearer and more accessible. However, the implementation already shows promise and is useful in its current state.

Bibliography

[1] Becker, B. A., and Quille, K. *Computer Science for Leaving Certificate*, 1 ed. Golden Key, Dublin, Ireland, 10 2020.

[2] Ben-Ari, M. Constructivism in computer science education. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 1998), SIGCSE '98, Association for Computing Machinery, p. 257–261.

[3] Buechley, L., Eisenberg, M., Catchen, J., and Crockett, A. The lilypad arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2008), CHI '08, Association for Computing Machinery, p. 423–432.

[4] Buechley, L., Eisenberg, M., and Elumeze, N. Towards a curriculum for electronic textiles in the high school classroom. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2007), ITiCSE '07, Association for Computing Machinery, p. 28–32.

[5] Edwards, S. H., Tilden, D. S., and Allevato, A. Pythy: improving the introductory python programming experience. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2014), SIGCSE '14, Association for Computing Machinery, p. 641–646.

[6] Greenberg, S., and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2001), UIST '01, Association for Computing Machinery, p. 209–218.

[7] Guzdial, M. A media computation course for non-majors. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*

(New York, NY, USA, 2003), ITiCSE '03, Association for Computing Machinery, p. 104–108.

[8] Hodges, S., Scott, J., Sentance, S., Miller, C., Villar, N., Schwiderski-Grosche, S., Hammil, K., and Johnston, S. .net gadgeteer: a new platform for k-12 computer science education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), SIGCSE '13, Association for Computing Machinery, p. 391–396.

[9] Hodges, S., Sentance, S., Finney, J., and Ball, T. Physical computing: A key element of modern computer science education. *Computer 53*, 4 (2020), 20–30.

[10] Horn, M. S., Crouser, R. J., and Bers, M. U. Tangible interaction and learning: the case for a hybrid approach. *Personal and Ubiquitous Computing 16*, 4 (2012), 379–389.

[11] Hoyles, C., Noss, R., and Adamson, R. Rethinking the microworld idea. *Journal of Educational Computing Research 27*, 1 (2002), 29–53.

[12] Lauwers, T., and Nourbakhsh, I. Designing the finch: Creating a robot aligned to computer science concepts. *Proceedings of the AAAI Conference on Artificial Intelligence 24*, 3 (Jul. 2010), 1902–1907.

[13] Lauwers, T., Nourbakhsh, I., and Hamner, E. Csbots: design and deployment of a robot designed for the cs1 classroom. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2009), SIGCSE '09, Association for Computing Machinery, p. 428–432.

[14] Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., and Werner, L. Computational thinking for youth in practice. *ACM Inroads 2*, 1 (feb 2011), 32–37.

[15] Malan, D. J., and Leitner, H. H. Scratch for budding computer scientists. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2007), SIGCSE '07, Association for Computing Machinery, p. 223–227.

[16] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. The scratch programming language and environment. *ACM Trans. Comput. Educ. 10*, 4 (nov 2010).

[17] Margolis, J., Fisher, A., and Miller, F. The anatomy of interest: Women in undergraduate computer science. *Women's Studies Quarterly 28*, 1/2 (2000), 104–127.

[18] Marshall, P. Do tangible interfaces enhance learning? In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (New York, NY, USA, 2007), TEI '07, Association for Computing Machinery, p. 163–170.

[19] Millner, A., and Baafi, E. Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In *Proceedings of the 10th International Conference on Interaction Design and Children* (New York, NY, USA, 2011), IDC '11, Association for Computing Machinery, p. 250–253.

[20] Petre, M., and Blackwell, A. F. Children as unwitting end-user programmers. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)* (2007), pp. 239–242.

[21] Rich, L., Perry, H., and Guzdial, M. A cs1 course designed to address interests of women. *SIGCSE Bull. 36*, 1 (mar 2004), 190–194.

[22] Salac, J., Thomas, C., Butler, C., Sanchez, A., and Franklin, D. Tipp&see: A learning strategy to guide students through use - modify scratch activities. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2020), SIGCSE '20, Association for Computing Machinery, p. 79–85.

[23] Sentance, S., and Schwiderski-Grosche, S. Challenge and creativity: using .net gadgeteer in schools. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education* (New York, NY, USA, 2012), WiPSCE '12, Association for Computing Machinery, p. 90–100.

[24] Sentance, S., Waite, J., Hodges, S., MacLeod, E., and Yeomans, L. "creating cool stuff": Pupils' experience of the bbc micro:bit. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2017), SIGCSE '17, Association for Computing Machinery, p. 531–536.

[25] Sentance, S., Waite, J., Yeomans, L., and MacLeod, E. Teaching with physical computing devices: the bbc micro:bit initiative. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (New York, NY, USA, 2017), WiPSCE '17, Association for Computing Machinery, p. 87–96.

[26] Stiller, E. Teaching programming using bricolage. *J. Comput. Sci. Coll. 24*, 6 (jun 2009), 35–42.

[27] Strong, G., and North, B. Pytch — an environment for bridging block and text programming styles (work in progress). In *Proceedings of the 16th Workshop in Primary and Secondary Computing Education* (New York, NY, USA, 2021), WiPSCE '21, Association for Computing Machinery.

[28] Tillberg, H. K., and Cohoon, J. M. Attracting women to the c. s. major. *Frontiers: A Journal of Women Studies 26*, 1 (2005), 126–140.