Implementing Scratch for CS Unplugged "Plugging it in"

Rchi Lugtu
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
rcl51@uclive.ac.nz

Tim Bell
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
tim.bell@canterbury.ac.nz

***Abstract*—Computer Science (CS) Unplugged is a free collection of activities created by the Computer Science Education Research Group (CSERG) used by teachers to teach computer science (CS) concepts away from a computer. Research shows that combining the Unplugged activities and computer programming gives students better self-efficacy in understanding CS concepts [1]. "Plugging it in" is a feature in CS Unplugged, which offers students Scratch and Python computer programming exercises based on the Unplugged activities. A basic system exists for Scratch exercises, which requires students to open a Scratch application in a separate window to do the exercises. This significantly decreases the usability and interactivity of the website. This paper reports on improvements to "Plugging it in" by implementing a more sophisticated system for the Scratch exercises. After doing extensive research, we found that the original goal was not feasible. Therefore, this solution uses Blockly, which has been adapted to look like Scratch. The new system includes an interactive editor with automatic question checking and browser program execution. The prototype was evaluated by three teachers who are part of CSERG to gather feedback to refine the system further. The same teachers evaluated the refined system using the think-aloud method, and it was received positively by them. We plan to further evaluate the system's effectiveness with more teachers in Digital Technology Teachers Aotearoa (DTTA) and students in New Zealand and all around the world.**

***Index Terms*—CS Unplugged, Python, Scratch, Block-based, Software Development, Computer Science Education Research**

I. Introduction

Computer Science Unplugged (CS Unplugged) is a website used by teachers to teach students computer science (CS) concepts without using a computer. It does this through kinaesthetic activities, games that use cards, strings, crayons, and lots of running around. This makes it a perfect tool to allow students to explore areas in CS without becoming experts in programming or owning a computer. CS Unplugged is available worldwide and can be accessed through the website csunplugged.org[1]. In addition, it is open-source, and so the activities are free and accessible for everyone.

There is a feature in CS Unplugged called "Plugging it in" which offers students programming exercises based on the Unplugged activities. The programming exercises are offered in both Python (text-based language) and Scratch (block-based language). These languages were chosen since Python is a great introductory language for students [2], and Scratch is a popular block-based language with over 73 million registered users[2]. A sophisticated system exists for Python exercises, allowing students to write their programs in an interactive editor and test it using an embedded interpreter. The Scratch, however, does not integrate well with the main exercises as it only displays the question and expected output to students. It does not have an interactive editor for students to create their programs within the website. As a result, students are forced to open the Scratch application[3] on a separate window, do the exercises there, and then compare the output of their program to the expected output displayed in the CS Unplugged website. This process is cumbersome and drastically decreases the website's usability and interactivity. Thus, the objective of this project is to implement a more sophisticated version of Scratch into "Plugging It In" to improve the overall experience of students when using the website.

Research shows the importance of combining the Unplugged activities with computer programming, as it fosters the computational thinking of children as well as increase their self-efficacy [1], [3]–[6]. Therefore, we believe it is important to invest time in looking at ways to continually improve "Plugging it in". We hope that more users will be encouraged to use this part of the website with this improved system.

This paper proposes a solution of implementing an improved system for the block-based exercises, which contains features such as an interactive block-based editor (Fig 1), browser program execution, automatic code checking, and submission feedback. The effectiveness of this system will then be evaluated by three experienced teachers who are part of CSERG. In the future, we plan to evaluate the system further by teachers in DTTA as well as students in classrooms.

II. Background and Objectives

A. Background

Implementing Scratch in CS Unplugged Plugging It In is an individual project created by the CSERG at the University

[1]https://csunplugged.org/

[2]https://scratch.mit.edu/statistics/

[3]https://scratch.mit.edu/projects/editor/?tutorial=getStarted

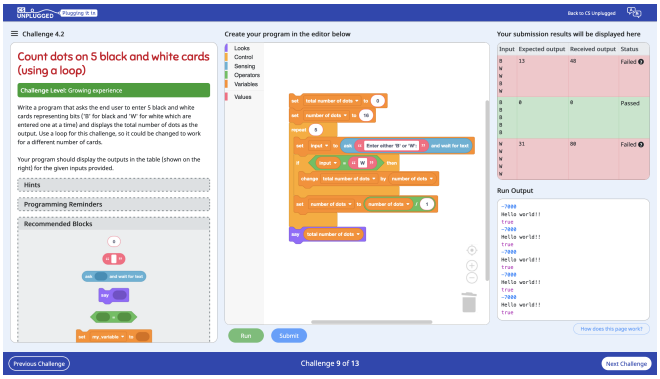

Fig. 1: Final version of the block-based system.

of Canterbury (UC). The parties involved in the project are as follows:

- Professor Tim Bell, the Academic Supervisor.
- Jack Morgan, the Technical Lead and Project Manager.
- Teachers who are part of CSERG.

Jack Morgan focuses more on supervising the software development aspect of the project. He deals more with the technical aspect of the project, while Tim Bell supervises the project as a whole. A weekly status meeting has been set up with the academic supervisor for a chance to ask questions and clarify any doubts.

Dana Lambert developed the existing system for Python as part of her SENG402 final year project in 2020. She improved the Python system's ease of use and interactivity by adding an interactive editor and automatic question checking to the Python programming challenges. This project is not directly linked to the project done last year (2020). However, we will take a similar approach to implement a new block-based system for the CS Unplugged website.

B. Objectives

The following work packages were created to ensure the different components of the project were clearly defined:

- Implement an interactive Scratch system into CS Unplugged.
- Evaluate how teachers use the system and how it could be made more effective during and after all implementation is finished.
- Implement testing and documentation for the added features.
- Develop connections to the topics content for teachers to access this new language.

This project aims to develop a block-based system to improve the current way of doing block-based programming challenges in "Plugging it in". This will be done by first researching Scratch to determine its suitability for CS Unplugged. If it is not suitable, then an alternative block-based language will be chosen instead. Next, a prototype system will be developed based on the chosen language. An evaluation will then be carried out by gathering feedback from teachers using an informal product review and think-aloud method to improve the system.

III. Related Work

The idea of CS Unplugged originated from the book called "Computer Science Unplugged... off-line activities and games for all ages" [7] written by Tim Bell, Ian Witten, and Mike Fellows. It started as an outreach program to help primary school students understand CS concepts without the overhead of learning programming first. [6]. The goal of the book was to teach CS topics through off-line activities (unplugged learning) and engage the audience without using a digital device. Some key principles of CS Unplugged involve learning through a constructivist approach, being highly kinaesthetic, using short and simple explanations, and having a sense of play or challenge [5]. The activities in CS Unplugged have been popularised and have since been used worldwide for classrooms, and out-of-school instruction [8]. Furthermore, this approach of learning CS concepts appeared in numerous CS Education book and was recommended in the 2003 ACM K-12 curriculum [5].

A. Plugged Learning

CS Unplugged is a great tool to engage students in learning CS concepts away from a computer. However, it is also important to link this new knowledge by going back to a computer and programming [9]. This way of learning is sometimes called "plugged" learning, and there is a feature in CS Unplugged called "Plugging It In" that offers this. "Plugging It In" is a collection of computer programming exercises for students to reiterate what they have learned from the activities in Unplugged. "Plugging It In" can be accessed through the CS Unplugged website[4].

1) Combination: There has been a growing debate between whether unplugged learning is better than plugged learning or whether a combination of both is better. A study was done of doing only unplugged learning to students with high anticipation of success. Results showed that students struggled with understanding the purpose of computer science even though they showed a good understanding of what it is [10]. A study [11] was conducted by Hermans and Aivaloglou to determine which method is more effective to start with: plugged or unplugged first. The experiment involved 35 elementary students. Half of the group were taught programming in Scratch for the first four weeks, while the other did pen and paper exercises. After these lessons, both groups used Scratch for another four weeks. Results showed no real difference in improvement with Scratch programming and their understanding of programming concepts. However, students who did unplugged lessons were more comfortable with their ability to understand CS concepts (i.e. better self-efficacy). This research suggests that programming exercises is more effective if used after unplugged learning has been introduced.

Another study [4] was done by Gardeli and Vosinakis that showed positive impacts on children's computational thinking

[4]https://csunplugged.org/en/plugging-it-in/

by combining unplugged and plugged learning. In this study, children worked in groups to program the behaviour of a popular computer game. The first part of this process involved designing a high-level version of students' algorithm using pen and paper. After this, they encode their solution as an executable program in a visual programming environment. Results showed that the pen and paper activity significantly helped students convert the algorithm they designed into a computer program. This is because students have already performed the algorithm in practice (with pen and paper). Therefore, they were able to record this knowledge in their memory. From this study, students seem to develop a better model in their brain about a particular concept if unplugged learning is used *first* before plugged learning.

B. Block-based Programming

"Plugging It In" offers programming challenges in Python (text-based language) and Scratch (block-based language). Block-based programming is another way of coding that utilises a drag-and-drop functionality with blocks to create programs. The most popular example of this is Scratch, created by MIT in 2007 to help introduce basic computer programming to children. Studies show that Scratch not only fosters the development of children's computational thinking, mathematical ideas, and problem-solving ability but also their interest in programming [12]–[14].

A five-week study [15] was conducted by Weintrop and Wilensky on using block-based and text-based languages to introduce programming to high school students. The findings from this study indicate that block-based students showed more significant learning gains and a higher level of interest in future computing courses. This suggests the importance of having block-based languages as an introductory approach to teaching programming. This is why implementing a block-based system like Scratch for "Plugging It In" is crucial, as it can significantly enhance children's knowledge and interest in programming.

IV. Initial Findings

Scratch is a popular block-based language with a big community supporting it worldwide. It has been ranked 22 in the TIOBE index as of June 2021[5] and is the only block-based language within the Top 25 languages. Currently, the problem with Scratch was that the team was unsure whether it had an API that we could hook into to have full access to its functionalities (i.e., interactive editor). Therefore, extensive research was done on whether Scratch could be implemented into CS Unplugged. First, I conducted a literature search from several areas such as WIPSCE, SIGCSE, Computer Science Education (tandfonline) journal, and Google Scholar to find studies that people have done regarding Scratch. Next, I explored discussion forums to determine whether people have done similar work that would be helpful for my project. From my research, I found some papers and discussion forums that showed many people are interested in Scratch. I also found projects where people have created modifications of the Scratch system, which extends the current base functionalities of Scratch. However, I was not able to find any information that directly related to my research project.

Next, since Scratch is open-source, I examined their repositories on GitHub[6] such as scratch-gui, scratch-vm, and scratch-www. However, I focused on the scratch-gui repository as it is responsible for handling Scratch's front-end and back-end logic. After thoroughly inspecting the repository, I did not find any API that would be helpful for the project. Without an API, intercepting the input/output of the programs in the Scratch editor would be challenging without directly modifying their codebase.

This approach is not ideal because modifying the Scratch code to work for "Plugging It In" has a high chance of being broken in the future if Scratch developers make a significant change in their codebase. The original solution of implementing Scratch into "Plugging It In" was not feasible. Therefore, different block-based languages were explored and evaluated to determine their suitability as a replacement for Scratch in "Plugging It In".

A. Evaluation of Different Options

We identified that the original solution of integrating the Scratch system in "Plugging It In" was simply not feasible. Therefore, I explored four other block-based languages to determine which is the most suitable replacement for Scratch. The following languages were chosen as they represent a variety of approaches and are popular. Only four were chosen due to the time available remaining on the project. For each option, I inspected their documentation, wiki, and discussion forums to check if an API exists or whether it can be implemented into "Plugging It In". In addition, I also downloaded their repository and examined it thoroughly. Lastly, I compared each option against the four primary requirements for a potential solution that my project supervisor and I have identified. These are listed as follows: maintainability, testing capability, similarities to Scratch, and whether it is open-source or not.

1) Blockly: Blockly is an open-source, client-side library created by Google[7] used by developers to add a visual code editor to web and mobile applications. It allows developers to create customisable and extensible blocks, making it a good alternative for Scratch. Blockly is flexible and maintainable because of its highly configurable and extensible functionalities. It also has the ability to convert block-based programs into executable code, making it easier to test students' programs.

2) Snap!: Snap*!* was developed by Brian Harvey and a former Scratch team member, Jens Mönig[8]. It is an HTML5 re-implementation of BYOB (Build Your Own Blocks), a modification of Scratch 1.4, which contains more advanced features. This means that the user interface looks similar to the old version of Scratch. One main advantage of this option

[5]https://www.tiobe.com/tiobe-index/

[6]https://github.com/LLK/

[7]https://developers.google.com/blockly/

[8]https://github.com/jmoenig/Snap

is that it has an API[9] that allows the Snap*!* editor to be accessed from the outside using broadcast messages. This API allows you to start scripts and access a project's global variables. Additionally, the API methods are maintained to work with future versions of Snap*!*, making it another suitable solution for Scratch.

3) Pencil Code: Pencil Code[10] is an open-source project that allows children to collaborate and create art, music, and games through code. After examining their repository[11], I discovered that Pencil Code does not have an API. This is the same problem with Scratch, and therefore, it would be difficult to intercept and test the input/output of programs. In addition, the blocks used in Pencil Code does not look like Scratch at all.

4) BlockPy: BlockPy is a block-based language that gives people a gentle introduction to Python, a text-based language.[12]. It is built using Blockly and uses Skulpt[13] to convert Python to JavaScript code, which can be run in a browser. One major disadvantage with BlockPy is that the blocks are based on Python syntax. This means that the blocks used in Blocky do not look like Scratch blocks at all. This can be difficult for students who have no prior experience with Python, and as a result, it could cause them to lose interest in "Plugging It in".

B. Chosen Solutions

After comparing the advantages and disadvantages of each option, it was determined that Snap*!* and Blockly proved to be the most viable solution for this project. A prototype was first implemented using Snap*!*. However, it was identified that the prototype forced users to put additional blocks (broadcast blocks) to their programs. This was not ideal, but it was the only way to intercept the input/output of the programs using the Snap*!* API. Therefore, we moved on to Blockly as the chosen solution for the project. A small proof of concept using Blockly was first developed, as shown in Appendix D. It was then enhanced further to create the prototype of the block-based system. This will be discussed in the next section.

V. Method and Project Management

A. Project Management

This project involved several people as follows. Professor Tim Bell is my academic supervisor and the product owner (PO) who supervised the project as a whole. Jack Morgan, the Technical Lead and Project Manager of CS Unplugged who helped me with development-related issues. The Computer Science Education Research Group (CSERG) provided valuable feedback that helped further improve the system. My task was to work with everyone to design and implement a working system that aligns with the project objectives.

This project used an iterative approach with a weekly meeting set up every Friday at 11:00 AM with the PO and Technical Lead. In this meeting, we evaluated my current progress and decided on tasks that needed to be done before the next meeting. A summary of the important points, design decisions, and completed tasks was recorded in a shared online document to ensure that everyone knows the current state of the project and its overall progress. In addition, there is also constant communication with everyone in Slack[14] to clarify any doubts I have related to the project.

B. Research Method

The research aspect of this project involves investigating and evaluating a suitable block-based system for CS Unplugged. We looked at the effectiveness of the new system specifically in terms of usability and interactivity. Regarding the design science cycle, the project will create an artifact of an improved block-based system into "Plugging It In". The project was evaluated by the teachers in the CSERG team through an informal product review and think-aloud session held in Zoom. The results of the evaluations identified helpful features that could be implemented in the future to refine the system further. The findings of this project and overall process are presented in this report, as well as in the project demonstration and poster.

C. Development Process

GitHub was used as the version control system during the development process. First, the CS Unplugged repository was forked, and a feature branch, *interactive-blockly-editor*, was created from the *develop* branch. When the feature branch is ready to be merged back into the develop branch, the test suite, containing automated tests and code style checks, must first pass. A pull request (PR) can then be created, which details the proposed changes. Another developer (most likely the Technical Lead) then reviews the PR and approves it if all checks pass. Approving the PR will merge all the changes from the PR and reflect it in the development website[15].

There are two iterations during the development process. For each iteration, a plan was conducted which involves reviewing the current implementation of "Plugging It In", gathering requirements, and adding tasks that were identified to the Trello board[16] as shown in Appendix B. From this, I created rough wireframes to communicate the user interface (UI) ideas quickly. The wireframes were evaluated by the PO and refined before being implemented. During the weekly meetings, the PO was updated regarding what I implemented. Then, we discussed the next steps moving forward. If any difficulties arise during development, the Technical Lead will immediately be notified to resolve the technical issue.

VI. Prototype Development

The goal of the prototype was to provide a working concept of a block-based system into "Plugging It In". It needed to have an interactive editor where users could create their programs and submit them to a test server. The server should

[9]https://github.com/jmoenig/Snap/blob/master/API.md
[10]https://pencilcode.net/
[11]https://github.com/PencilCode/pencilcode
[12]https://think.cs.vt.edu/blockpy/
[13]https://skulpt.org/
[14]https://slack.com
[15]https://cs-unplugged-dev.csse.canterbury.ac.nz/en/
[16]https://trello.com/b/DDBpJV9U/seng402-blockly-system

test their program through a series of built-in test cases to determine whether they have solved the exercise or not. It should also display helpful and meaningful feedback from the results they receive.

This section will cover some of the critical design decisions made for both the UI and implementation of the prototype. Following that, a high-level overview of the architecture, the technological stack, and the primary features implemented will be discussed. Figure 2 shows the prototype created for the block-based system.

Fig. 2: Prototype of the block-based system.

A. Design

1) Page Layout: Wireframes were created to explore several design options as shown in Appendix C. After showing the wireframes and discussing it with the PO, we decided to use the full-screen three-column layout, closely following the Python system's design. The first column contains information about the challenge, the middle column is the interactive editor, and the third column contains the results table. This was chosen to increase the internal consistency of "Plugging It In" by having the same layout as the Python programming challenge page. This follows one of Nielsen's 10 Usability Heuristics[17] which is being consistent with the UI. Having consistent UI allows any prior learning to be brought over from the Python system and into the new block-based system. One disadvantage of this layout is that there is not much space in the editor for users to create their programs.

Other design alternatives were considered, such as the full-screen two-column layout as shown in Appendix C. The main advantage of this is that it provides users more space in the editor to create their programs. However, we decided not to use this design as it requires users to familiarise themselves with the page layout and where the elements are since it has a different layout to the Python system. Therefore, we used the three-column layout instead of the two-column layout. To compensate for the narrow editor, we made the third column (containing the results table) narrower to give the middle column (containing the block-based editor) space.

2) Custom Blocks: Only a subset of Scratch blocks[18] was replicated as some were not useful in the context of this project (e.g., Motion and Sounds blocks). Therefore, only the Operators, Variables, and some of the Looks, Sensing, and Control blocks, were replicated. This was discussed with the PO, and we were concerned this might be confusing. Therefore, I talked to the teachers in CSERG, and they said that showing a subset of blocks would be a good idea.

3) Run button and Recommended Blocks: The Run button used to execute users' programs in the browser was positioned beside the Check button to make the website look consistent. The Recommended Blocks was placed alongside the Hints section. However, the images of the blocks were not displayed yet in this version of the system.

B. Implementation

CS Unplugged is an open-source web application on GitHub[19] built using Django[20], a Python web framework. The technologies used in the front-end are Javascript (JS), SASS (Syntactically Awesome Style Sheets), and Bootstrap, while a Postgres database is used for data storage. A Django project consists of several Django applications that are responsible for specific features in a website (e.g., "Plugging It In" is an application in the CS Unplugged website). CS Unplugged uses a Model-View-Controller (MVC) architecture, which separates the application's business logic and presentation details. A Django application usually contains a model which is the definitive source of information about a particular data. It generally maps to a single database table.

An application also contains views. Views interact with database models through a feature called Object-Relational Mapper (ORM), allowing a developer to write Python code instead of SQL to perform database operations. A view can also pass context data to templates so that the browser can render it. A template is a Python text document (similar to HTML) that uses the Django template language to present information to the user. Views and templates can be linked together to define endpoints using Django URL files.

CS Unplugged uses Docker to run the system, both on a local machine for development and when deployed to production. The different parts of the system (i.e., jobe server, Postgres database, Django project, node, and elasticsearch) are stored in different containers through a process called containerisation. Using Docker allows a developer to set up the website locally without installing different technologies/dependencies in their machine to run the application.

1) Development Setup and Workflow: Visual Studio Code and macOS were used as the chosen editor and operating system for development. Docker was installed, and it handled all the required dependencies to run the CS Unplugged project locally. Most of the development for the front-end and back-end of the block-based system used the existing architecture of the Python system in the plugging_it_in Django application.

[17] https://uxgorilla.com/nielsens-heuristics/

[18] https://en.scratch-wiki.info/wiki/Blocks

[19] https://github.com/uccser/cs-unplugged

[20] https://www.djangoproject.com/

2) Adding the Block-based Editor: The plugging_it_in application handles all the components to run "Plugging It In", such as the programming challenge page, which displays the Python editor. This page was modified to display either the Python or block-based editor if the URL path ends with '/python' or '/block-based', respectively. The views and models responsible for displaying the Python editor were also modified to accommodate the block-based editor. A new Django template file was added to represent an information page showing the differences between the block-based language and Scratch. JS files were also created to handle the front-end logic.

3) Automatic Code Testing: Blockly has functionality that allows blocks (either custom or default Blockly blocks) to be converted into executable code using code generators[21]. A code generator is a function that instructs Blockly on how a specific block can be converted into a programming language. It supports the following languages: JavaScript, PHP, Python, Lua, and Dart. Currently, the "Plugging It In" Python system sends users code to a back-end interpreter called Jobe and retrieves their results against the test cases. Jobe[22] (Job Engine) is a server developed by Richard Lobb that runs small programming jobs in a variety of programming languages. With this information, we achieved automatic code testing by creating Python code generators for each custom block. This converts the blocks into Python code, and then we used the existing functionality of the Python system to send the code to Jobe and retrieve the results.

4) Browser Program Execution: Implementing this functionality was similar to implementing automatic code testing. It also uses code generators, but instead of creating Python ones, we created JavaScript code generators. This converts the blocks into JavaScript code (in string format), which can then be executed using the built-in eval() function in JavaScript. The eval() function treats a string as if it were an expression and returns the result. With this, the Run button was created, and we linked the logic to this button. Therefore, whenever the user clicks the button, it will call a function that will execute their program in the browser.

5) Custom Blocks Creation: As mentioned earlier, the objective of the prototype was to provide a working concept of a block-based system in "Plugging It In". A small proof of concept using Blockly was first developed, as shown in Appendix D. However, the system uses the default blocks of Blockly. The default blocks did not look similar to Scratch. Hence, I created the custom blocks and made them look like Scratch.

This was done using the Blockly Developer Tools[23], a graphical user interface (GUI) tool that automates parts of the Blockly configuration process, which includes creating custom blocks, building the toolbox, and configuring the Blockly workspace. The tool allows developers to create block definitions and code generators for custom blocks. Block definitions can either be in JSON or JavaScript, and it tells Blockly how the blocks are rendered in the browser. Block definitions can be further extended by applying themes to make them look like Scratch. However, this was not done as we only focussed on the functionality at this stage. The block definitions and code generators for the custom blocks were abstracted out in a separate JS file (custom-blockly-blocks.js).

VII. Prototype Evaluation

A. Block-based System versus Scratch

Following the development of the prototype, several differences between the block-based system and Scratch can be identified. This is due to the technical issues we experienced relating to how Blockly was built as well as the time available for development, making implementation challenging. However, as a result, this paved the way for creating blocks to help students transition from block-based to text-based programming. Since, in the real world, text-based languages are what is mainly used to develop sophisticated applications.

1) Values Blocks and Shadow Blocks: One major difference that Scratch users will recognise is the Values blocks, which relate to different programming data types (i.e., integer/float, string, and boolean), as shown in Fig 3. In Scratch, some blocks like the math Addition block can accept string, boolean, and numbers as input. As a result, distinguishing the type of input for the Addition block would be complicated, and although it can be replicated, it would be difficult to do so given the available time. Therefore, we created the Values blocks as a solution to this problem. This solution uses Blockly's data type blocks (i.e., number, text, and boolean block) by grouping them in a new category called "Values". This category provides a centralised location for the new blocks.

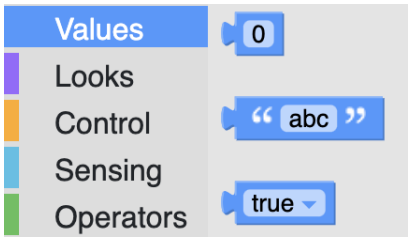

Fig. 3: Values Category containing different data type blocks.

Another solution to mediate the problem was by utilising Blockly's Shadow blocks[24] as shown in Fig 4. Shadow blocks are blocks that can be embedded within different blocks. It cannot be removed or thrown away. However, if users want to replace it, they can simply overwrite it by putting another block on top of it. The shadow block will not pop out like a regular block.

[21] https://developers.google.com/blockly/guides/create-custom-blocks/generating-code

[22] https://github.com/trampgeek/jobe

[23] https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

[24] https://groups.google.com/g/blockly/c/bXe4iEaVSao

2) Ask and Wait Block for Number/Text: Another difference is the Ask and Wait blocks - Ask and Wait for Text and Ask and Wait for Number, which outputs either a string or number, respectively. These two blocks were created due to the same problem discussed in the previous section. In Scratch, the Ask and Wait block automatically stores the user input in the special answer[25] variable. However, in the block-based system, the Ask and Wait blocks only output the user input value. This means users must manually store the output of the two blocks in a variable to save the user input. This might confuse users, especially if they are used to using Scratch. However, we hope that the addition of the two new blocks will reinforce, alongside the Values blocks, the concept of different data types in programming and further bridge the gap between block-based and text-based programming.

3) Running the Code: In Scratch, the 'When Green Flag is clicked' block is used to start users programs. However, it was decided not to create this block since it does not significantly benefit the block-based system. Instead, the Run button was created to allow users to execute their program in the workspace (i.e., the area where users create their program) before submitting their program.

4) Displaying the Output: The block-based system currently uses JavaScript's alert() function to display the Say block's output. This is quite different to Scratch because, in Scratch, the output is displayed in the speech bubble of the Scratch Cat[26]. A similar feature can be implemented, however, we focussed on the functionality of the block-based system at this stage, and therefore it was not implemented.

5) Block Shapes: The shapes of the blocks were also different compared to the Scratch blocks. However, this can easily be configured by either using the Zelos renderer[27] or by using a Blockly theme[28]. Block shapes do not affect the functionality of the prototype, and therefore, it was not considered during this stage.

B. Product Review Evaluation

The prototype was evaluated through an informal product review from three experienced teachers who are part of the CSERG team. Due to the unexpected increase to level four lockdown in New Zealand, the evaluation was conducted via Zoom. This made the evaluation process slightly challenging as technical difficulties were encountered while demonstrating the prototype. However, the teachers were able to give valuable feedback throughout the evaluation process, which helped refine the system. In this evaluation, an individual Zoom session was held for each of the three teachers to ensure their thoughts were of their own and not influenced by the other teachers. An advantage of this evaluation was that it allowed me to get quick feedback on the system and find problems early on before spending lots of time on development.

[25]https://en.scratch-wiki.info/wiki/Answer_*block*
[26]https://en.scratch-wiki.info/wiki/Scratch_Cat
[27]https://blocklycodelabs.dev/
[28]https://developers.google.com/blockly/guides/configure/web/themes

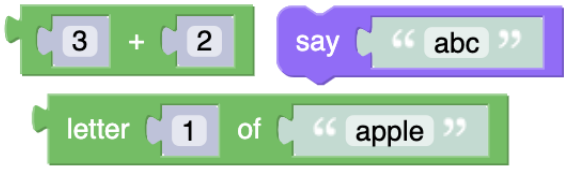

Fig. 4: Shadow blocks predefined in the Operator blocks and Say block.

In the session, a brief overview of the project and its objectives were explained, which was to make the system look like Scratch. The prototype was then demonstrated to the teachers to show how it works in "Plugging It In". In addition, the differences between the block-based system and Scratch were mentioned. Subsection VII-A provides more information about the differences between the two. Since each teacher had different backgrounds with teaching block-based programming, particularly Scratch, they were able to identify specific problems in the system that could negatively affect the user experience when completing block-based programming questions in "Plugging It In". After conducting the evaluation, I was able to categorise the problems identified into five main categories.

1) Data Type Blocks: One major problem they had at the start of the session was understanding what the Values Blocks were for. The teachers understood the principle, but because they had never seen it before, they were unable to connect how to program with the blocks. Furthermore, some blocks such as the Operator blocks and Say block already contained predefined Shadow blocks in which users can directly type their input in, as shown in Figure 4. As a result, they did not get the chance to use the Values blocks, preventing them from making the connection of how to program with it. A solution recommended was to replace the Shadow blocks with the actual Values blocks, as this would demonstrate how the blocks are used. Another suggestion was to provide a Programming Reminders section similar to the Python system. Not only does it provide examples of how to use the blocks, but it also teaches students that programming concepts can be applied to any language; all they have to do is learn the syntax.

2) Program Output: Another major issue the teachers identified was how their program's output was being displayed as a JavaScript alert box. They found that it makes it appear disconnected from the block-based system and obscures the user's code in the workspace, making it difficult to track where their program is currently being executed. It also makes it challenging to keep track of multiple outputs because the user only sees one output at a time. They also must manually click the OK button to move on to the next one. This can give users a negative experience, especially if they keep getting the questions wrong, as they would have to repeatedly click the OK button to see the rest of the output. Several suggestions were made to improve this by displaying the program output underneath the results table. Another suggestion was

to display program outputs in a shell-like interface similar to how text-based languages outputs are displayed (e.g., Python, JavaScript, or C).

3) Run and Check Buttons: It was also observed that clicking the Run button to execute their code was quite confusing since they were used to clicking the green flag button to start their program. Furthermore, they instinctively looked for the 'When Green Flag Clicked' block[29] to put at the top of their program. We expected this kind of behaviour since Scratch users are accustomed to seeing these key elements. A recommendation was suggested by changing the colour of the Run button to the same colour as Scratch's green flag. This will help Scratch users make that correlation with Scratch's green flag that the Run button executes their program.

The teachers also found it hard to differentiate between the Check and Run buttons. A suggestion was given to change the button's text from 'Check' to 'Submit' to make the two buttons distinct and have good affordance. Good affordance is important as it gives visual signals and psychological shortcuts that assist users to recognize the tasks they may perform on an application, in this case, the Run and Submit buttons.

4) Ask and Wait Block: Teachers were also confused with the 'Ask and Wait for Number/Text' blocks being connected sideways. Normally, the 'Ask and Wait' block in Scratch automatically sets the output to a special *answer* block. This means that the 'Ask and Wait' block does not have to be connected sideways. However, they like the idea that it slowly transitions students to text-based programming since they are forced to manually set the output to a variable, rather than letting the system automatically do it for them, like in Scratch. Furthermore, the teachers also liked the idea of having two separate blocks for the different data types, i.e., number and text, as it is starting to guide students to the idea that there are different data types in programming. A suggestion was made to provide examples in the Programming Reminders section on how the blocks can be used. This will avoid confusion when users see the blocks for the first time.

5) Tooltip: Finally, the tooltip proved to be helpful. However, the teachers were unaware that it existed until they were informed about it. As a result, one suggestion was to find a way to inform users that it exists and how they can access it.

VIII. Beta Development

The goal of the beta version was to enhance the prototype and create a more refined block-based system addressing the problems identified from the initial evaluation. Several recommendations were received and discussed with the product owner, Tim Bell, to determine their suitability for "Plugging It In". The major features added in the beta version were

- Programming syntax reminders
- Recommended blocks
- Improved blocks
- Shell-like interface that displays the output of the programs

[29]https://en.scratch-wiki.info/wiki/Green_Flag

Fig. 5: Beta version of the block-based system.

Figure 5 and Appendix E shows the beta version of the block-based system.

A. Design

1) Shell Interface: Based on the feedback gathered from the initial evaluation, it was identified that finding a better method of displaying the output of a program to users is important. A shell-like interface was added and placed at the bottom of the results table as it was the most optimal position given the current page layout. A title was added above the shell interface labelled "Run Output" to remind users that clicking the Run button display their program output there.

Initially, the outputs displayed in the shell interface were just plain black text. However, upon discussing with the PO, we determined that this was a good opportunity to further reinforce the concepts of different data types in programming. Therefore, the colour of each data type (i.e., string, number, and boolean) in the interface was changed to black, green, and purple. Further, the border colour of the shell interface and results table were changed to green and blue respectively, to make a strong correlation to the Run and Submit buttons.

2) Programming Reminders and Recommended Blocks: The teachers found the Ask and Wait for number/text blocks confusing at the start of the evaluation. It was only after I demonstrated how the blocks were used was when they understood how it works. Therefore, the Programming Reminders section was added, providing a reference point for users to see how the new blocks are used.

On the other hand, the Recommended Blocks were added to support students struggling with the programming exercises. The Programming Reminders and Recommended Blocks sections were added alongside the Hints section to group all the support content in one area.

B. Implementation

1) Improved Custom Blocks: A renderer is a functionality Blockly provides which translates a block model to SVG. It takes a block definition (in either JSON or JS) and dynamically lays out a block on the page. One type of renderer Blockly offers is the Zelos renderer, which makes the shapes of the blocks look like Scratch. Therefore, the Zelos renderer was

added to the custom blocks. I also changed the colours of the Values blocks to be the same colour as Scratch's My Blocks[30].

In Scratch, users can create custom blocks through My Blocks. Any custom blocks that users create have a colour of salmon pink. Therefore, this colour was used in the Values blocks since these blocks are new in Scratch, and so it gives users the illusion that they are just custom blocks in Scratch. The Zelos renderer combined with the colours of Scratch makes the custom blocks significantly resemble Scratch blocks, as shown in Figure 6.

Fig. 6: Custom blocks with Zelos renderer.

2) Recommended Blocks and Programming Reminders: The Recommended blocks are like Parson's problem[31] where users are provided with all the lines of code required to solve the problem, but with the lines jumbled. In this case, the idea was to display images of pre-defined blocks, and all the students had to do was recreate the blocks and figure out how to connect them to solve the challenge. This idea was taken from the Scratch system in "Plugging It In". It uses Verto[32], a program that uses a special Scratch image creator to render texts into images of Scratch blocks.

Upon reading Verto's documentation, I discovered that there was no version to do this for the blocks in Blockly. Therefore, one solution was to take screenshots of all the pre-defined recommended blocks for all the programming exercises. However, this was not ideal as it would use up a significant amount of storage to store all the images. Therefore, a workaround was to take screenshots of the individual default blocks. This solution uses less storage because we can reuse the images of the default blocks and display a subset of them depending on the programming exercise. The downside of this is that users would still need to figure out how to solve the exercise as it does not directly display the exact blocks they need to use.

The examples for the Programming Reminders section closely follow the examples from the Python system. This was done to keep consistency so that students will still be familiar with the examples provided in the Programming Reminders if they switch from Python to the block-based exercises. I used the same approach to display the images, which was to take a screenshot of each programming example I created.

3) Shell Interface: The shell interface was implemented by modifying the Say block's JavaScript code generator, which displays the program outputs when run in the browser. The JS code generator takes the reference of the shell interface in the front-end using document.querySelector() and dynamically modifies it to add the output. For each output, a span tag with a CSS class (console.default, console-number, console-boolean) is added to it depending on what type of data it is (string, number or boolean). A rule is added for each CSS class to determine the colour used when it is displayed in the shell interface.

4) Tutorial: Since the block-based system uses the existing architecture of the Python system, the "How does this page work" button were also set up to work for the new system. This button goes through the different parts of the system explaining what they are and how they work. This is helpful for first time users who are confused on how to use the system.

IX. Beta Evaluation

The second version of the system was evaluated by the same teachers from the first evaluation using the think-aloud method. Overall, the teachers received the system positively, and only a few minor problems have been identified.

A. Think Aloud Evaluation

The think-aloud method[33] is a strategy that requires users to verbalise their thoughts out loud while performing a set of tasks on a system. This method was used to gain an insight into what aspects of the system were confusing for the users.

An individual Zoom session was held for each of the three teachers, similar to how I did the initial evaluation. However, in this evaluation, the teachers were able to use the system for the first time. In the session, I gave a brief explanation of how the think-aloud evaluation works. The teachers were also encouraged to look at the Programming Reminders section and click on the "How does this page work?" button to help them familiarise themselves with how the block-based system works. Next, they were instructed to solve a set of programming exercises in "Plugging It In" with increasing difficulty. If the teachers were stuck, then I guided them on how they could solve the exercise. During the evaluation, I wrote down notes on areas they were unsure about and categorised them.

1) Recommended Blocks: All teachers found the Recommended Blocks useful. However, one of the teachers had a problem with consistently attempting to drag the images of the blocks from the Recommended Blocks section into the workspace. This behaviour occurred multiple times throughout the evaluation.

2) Shell interface and Results table: During the evaluation, one of the exercises required the teachers to create a program that outputs a number. When they clicked the Run button, all teachers were under the impression that they had already solved the exercise, when in reality, they had not. This was because the numbers were coloured green when outputted in the shell interface. Furthermore, the border of the shell interface was also green, giving the appearance that they got the

[30] https://en.scratch-wiki.info/wiki/My_Blocks
[31] https://en.wikipedia.org/wiki/Parsons_problems
[32] https://verto.readthedocs.io/en/latest/
[33] https://pidoco.com/en/help/ux/think-aloud

exercise correct. A suggestion was made not to use the green colour when a number is displayed in the interface. Overall, all teachers found the shell interface extremely useful and significant improvement from the previous way of displaying program outputs. They also found the different colours for the different data types unique and useful.

3) Values blocks: Like the initial evaluation, the teachers found the Values blocks confusing as they had not come across them before. However, the Programming reminders section gave clarity on how the blocks are used.

X. Outcome

From the think-aloud evaluation, the results gathered were immediately discussed with the PO. Refinements were made to the system, such as changing the colour of the number in the shell interface to blue and removing the border colours from the shell interface and the results table. The download button was also removed as it was irrelevant in the context of the system.

After performing manual tests and writing automated tests for the new system, a PR was created to merge the new system into the development website of CS Unplugged. Screenshots of the outcome of the project are shown in Appendix F

1) Limitation: There were several limitations identified with the development and evaluation process of the project. Firstly, only three teachers evaluated the system, which is not enough to represent the population accurately. CS Unplugged is used worldwide, and most of its users are actually from the US. Small sample size affects the reliability of the evaluation's results as it leads to a higher variability, which may lead to bias. As a result, not all problems would have been identified in the system. Furthermore, the results gathered were from the perspective of the teachers and not the students. Therefore the feedback received may not truly represent the target audience since the system is intended for students to use following on from the Unplugged activities.

Secondly, the teachers already had an idea of how the system works from the first evaluation. Therefore, during the second evaluation, they would already be slightly familiar with the system. This meant that the results gathered from the think-aloud evaluation may not be reliable as it was not their first time seeing the system.

The third limitation relates to the software application itself. Programs behave differently when being run in the browser and when being tested by the testing server. This is because the blocks are converted into JavaScript when it is run in the browser but converted to Python when being tested. Python and JavaScript behave differently, and therefore the programs behave slightly differently as well. Furthermore, I may have over-engineered the system since students are not penalised when they submit their programs. Therefore, the program execution in the browser could be removed to solve this inconsistent behaviour problem.

2) Reflection: Overall, this project has helped me gain confidence in tackling complex projects that involve rigorous research and working alongside people with varying backgrounds to develop and evaluate a software application. It also gave me the opportunity to do a think-aloud evaluation which I have learnt about in previous courses at university but never was able to apply it.

Overall, it was interesting to learn about the world of block-based programming and its importance to education in the computer science field. It was a fantastic experience working alongside an amazing team which helped me develop the essential skills a software engineer should have.

XI. Conclusion

CS Unplugged has a feature called "Plugging it in" that offers students programming exercises in Python and Scratch following on from the Unplugged activities. A basic system exists for Scratch exercises, which forces students to create their programs from outside the website, decreasing the usability and interactivity of the website. A new block-based system in "Plugging It In" was designed, developed and evaluated. The system contains an interactive editor, browser program execution, automatic program checking, program persistence, and support content to address the previously mentioned problems. It was evaluated by experienced teachers in the CSERG team, which helped further improve this new feature. We plan to further evaluate this system with more participants to determine its effectiveness in "Plugging It In". The new block-based system will be merged in the development website of CS Unplugged.

XII. Future Work

Based on the feedback, a list of actionable items were created to improve the new system further. First, more research could be done into Blockly's API on whether it is possible to fully replicate Scratch's functionalities that could not be recreated in this project. This includes implementing the "Ask and Wait" block in Scratch, where the output is automatically stored in the special Answer variable. The custom blocks could also be modified to accept different data types as input, removing the need for the Values blocks. In terms of getting user input with the 'Ask and Wait" block, an enhancement could be made to make an input text field appear at the bottom of the shell interface. This will remove the obtrusive pop up which prompts users for input. Lastly, further evaluation could be made with more participants (teachers and students) in New Zealand and all around the world. This would give us better insight into the effectiveness of the new block-based system in "Plugging It In".

Acknowledgment

A massive thanks to Tim Bell for providing support and guidance throughout all the project phases and Jack Morgan for the technical support during development. Thank you also to the three teachers in the CSERG team for providing helpful feedback, which significantly improved the system.

REFERENCES

[1] Felienne Hermans and Efthimia Aivaloglou. To Scratch or Not to Scratch?: A Controlled Experiment Comparing Plugged First and Unplugged First Programming Lessons. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, WiPSCE '17, pages 49–56, New York, NY, USA, 2017. ACM.

[2] A Bogdanchikov, M Zhaparov, and R Suliyev. Python to learn programming. *Journal of Physics: Conference Series*, 423:012027, April 2013.

[3] Time to plug back in? The role of "unplugged" computing in primary schools - University of Chichester ChiPrints Repository.

[4] Anna Gardeli and Spyros Vosinakis. Creating the computer player: an engaging and collaborative approach to introduce computational thinking by combining 'unplugged' activities with visual programming. *Italian Journal of Educational Technology*, 25(2):36–50, August 2017. Number: 2.

[5] Tim Bell and Jan Vahrenhold. CS Unplugged—How Is It Used, and Does It Work? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 497–521. Springer International Publishing, Cham, 2018.

[6] Tim Bell, Frances Rosamond, and Nancy Casey. Computer Science Unplugged and Related Projects in Math and Computer Science Popularization. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 398–456. Springer, Berlin, Heidelberg, 2012.

[7] Tim Bell and Ian H Witten. Computer Science Unplugged . . . off-line activities and games for all ages. page 240, 1998.

[8] Caitlin Duncan and Tim Bell. A Pilot Computer Science and Programming Course for Primary School Students. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE '15, pages 39–48, New York, NY, USA, November 2015. Association for Computing Machinery.

[9] Rivka Taub, Mordechai Ben-Ari, and Michal Armoni. The effect of CS unplugged on middle-school students' views of CS. *ACM SIGCSE Bulletin*, 41(3):99–103, July 2009.

[10] Yvon Feaster, Luke Segars, Sally K. Wahba, and Jason O. Hallstrom. Teaching CS unplugged in the high school (with limited success). In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, ITiCSE '11, pages 248–252, New York, NY, USA, June 2011. Association for Computing Machinery.

[11] Felienne Hermans. To Scratch or not to Scratch? page 8, 2017.

[12] Luis Alberto Calao, J. Moreno-León, Heidy Ester Correa, and Gregorio Robles. Developing Mathematical Thinking with Scratch. In Gráinne Conole, Tomaž Klobučar, Christoph Rensing, Johannes Konert, and Elise Lavoué, editors, *Design for Teaching and Learning in a Networked World*, Lecture Notes in Computer Science, pages 17–27, Cham, 2015. Springer International Publishing.

[13] Full article: Computational thinking and mathematics using Scratch: an experiment with sixth-grade students.

[14] José Antonio Rodríguez-Martínez, José Antonio González-Calero, and José Manuel Sáez-López. Computational thinking and mathematics using Scratch: an experiment with sixth-grade students. *Interactive Learning Environments*, 28(3):316–327, April 2020.

[15] David Weintrop and Uri Wilensky. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, 18(1):3:1–3:25, October 2017.

Appendix A: Project Management Evidence

Screenshots of the Trello board I used to help track my progress throughout the project.

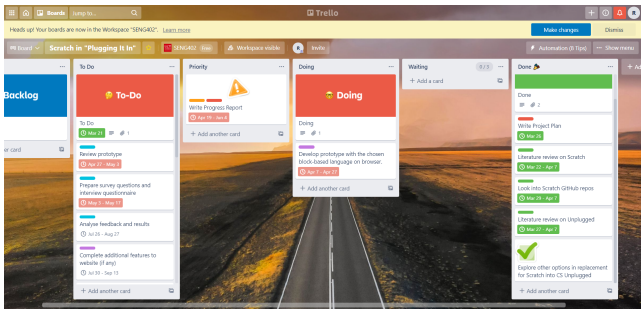

Fig. 7: Trello board progress in mid-year.

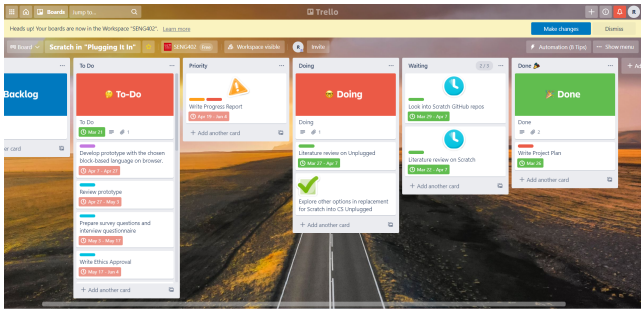

Fig. 8: Trello board progress at the start of the year.

Appendix B: Development Process Evidence

Screenshot of the Trello board I used to help track my development process when implementing the new system into "Plugging It In".

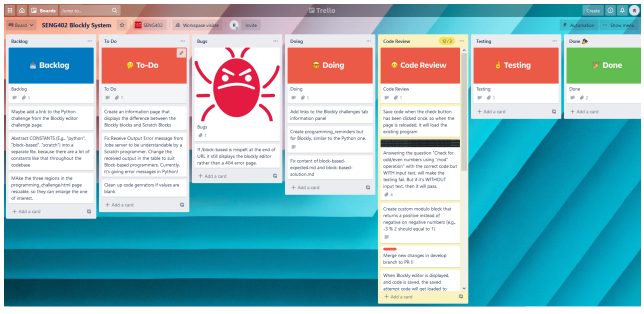

Fig. 9: Prototype Development Trello Board.

Appendix C: Prototype Wireframes and Mockups

Screenshots of the wireframes I created using Mockflow's Wireframe tool to display the different kinds of page layout I considered when designing the new system. The different layouts I considered were the two-column layout and the three-column layout.

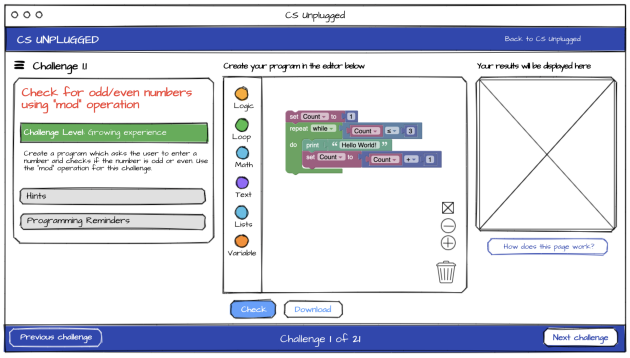

Fig. 10: Prototype wireframe three-column layout

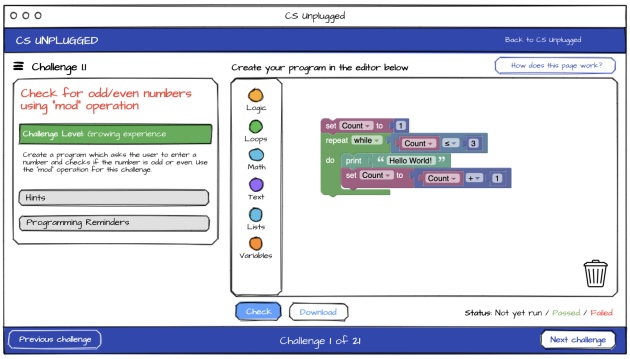

Fig. 11: Prototype wireframe two-column layout

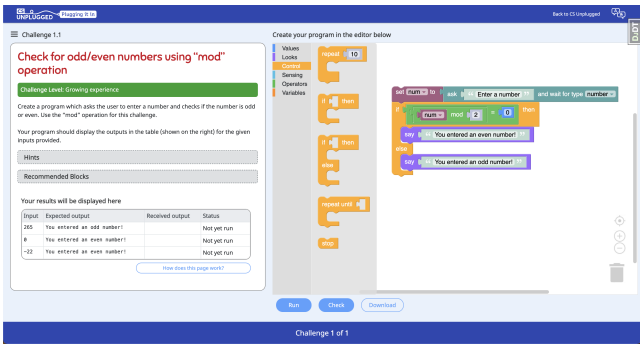

Fig. 12: Prototype two-column layout first design alternative.

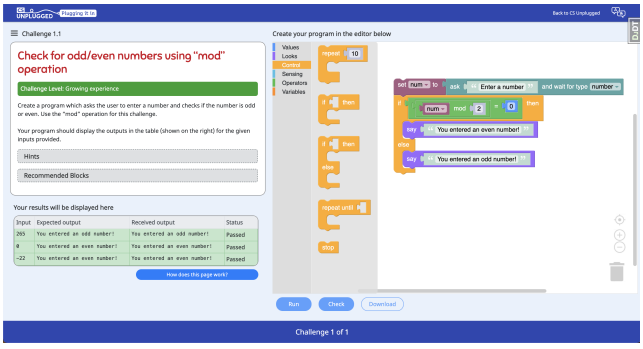

Fig. 13: Prototype two-column layout second design alternative.

Appendix D: Prototype Development

The proof of concept I developed using Blockly to determine its suitability in "Plugging It In" before fully implementing the block-based system.

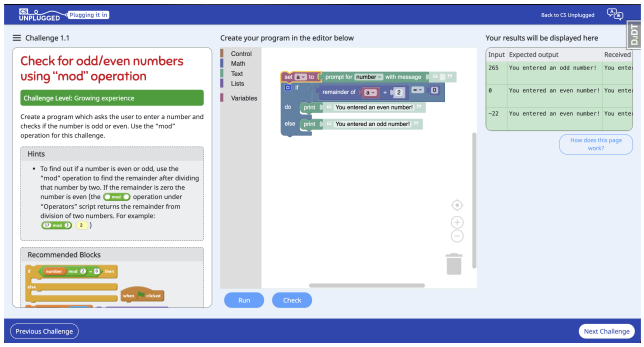

Fig. 14: Proof of concept containing the default blocks of Blockly.

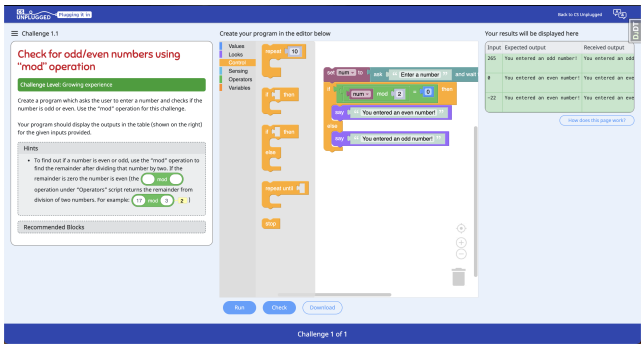

Fig. 15: Prototype version of the block-based system.

Appendix E: Beta Development

A screenshot of the refined system after the initial evaluation. The refined system includes a shell-like interface to display the outputs of users' programs and visual improvements to the custom blocks.

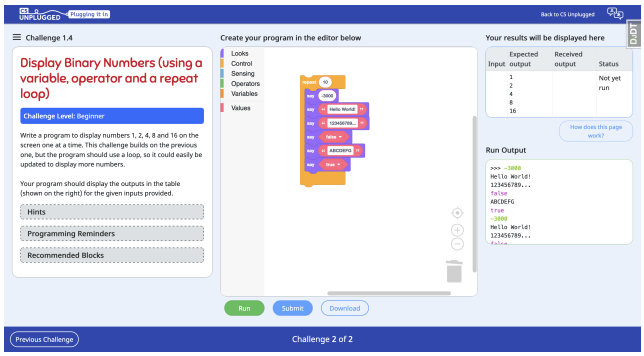

Fig. 16: Beta version of the block-based system.

Appendix F: Final Outcome

Screenshots of the final version of the new block-based system in "Plugging It In". It displays the page where users solve the programming exercises, a page that explains the differences between the new system and Scratch, and finally, the home screen of "Plugging It In" which shows the new block-based language.

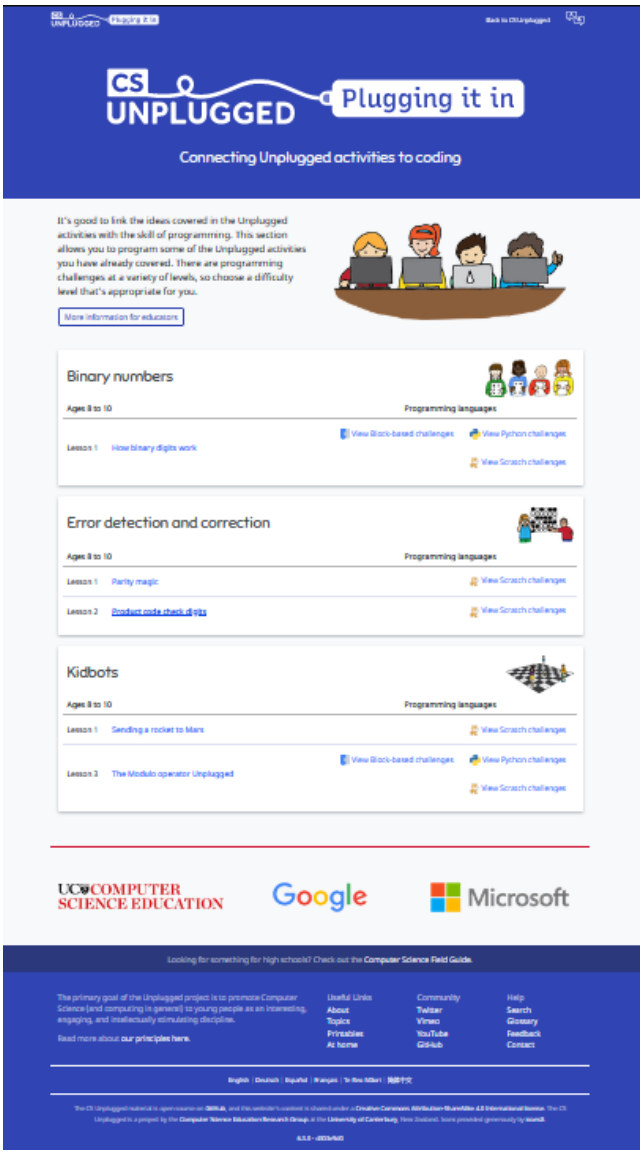

Fig. 17: New language added to Plugging It In.

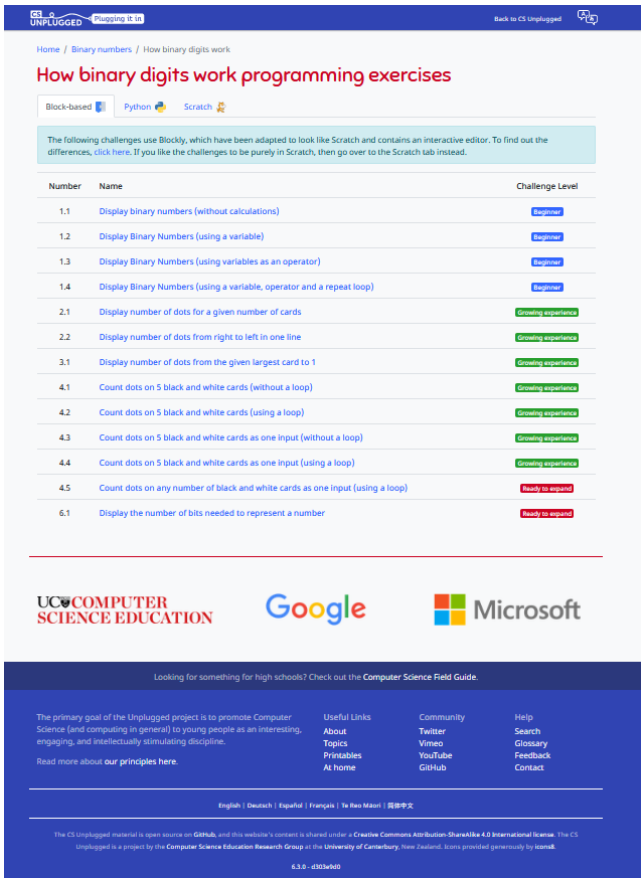

Fig. 18: List of programming exercises for the new language.

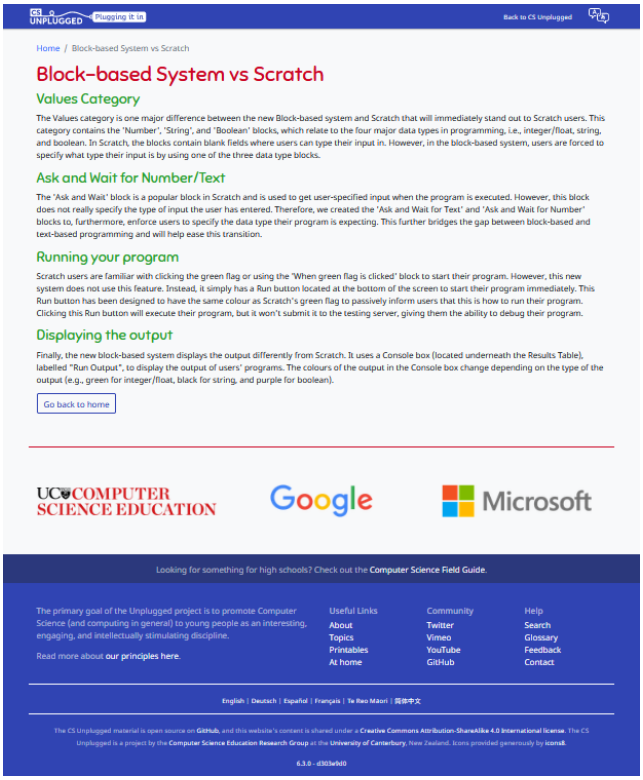

Fig. 19: Information page describing the differences between the new block-based system and Scratch.

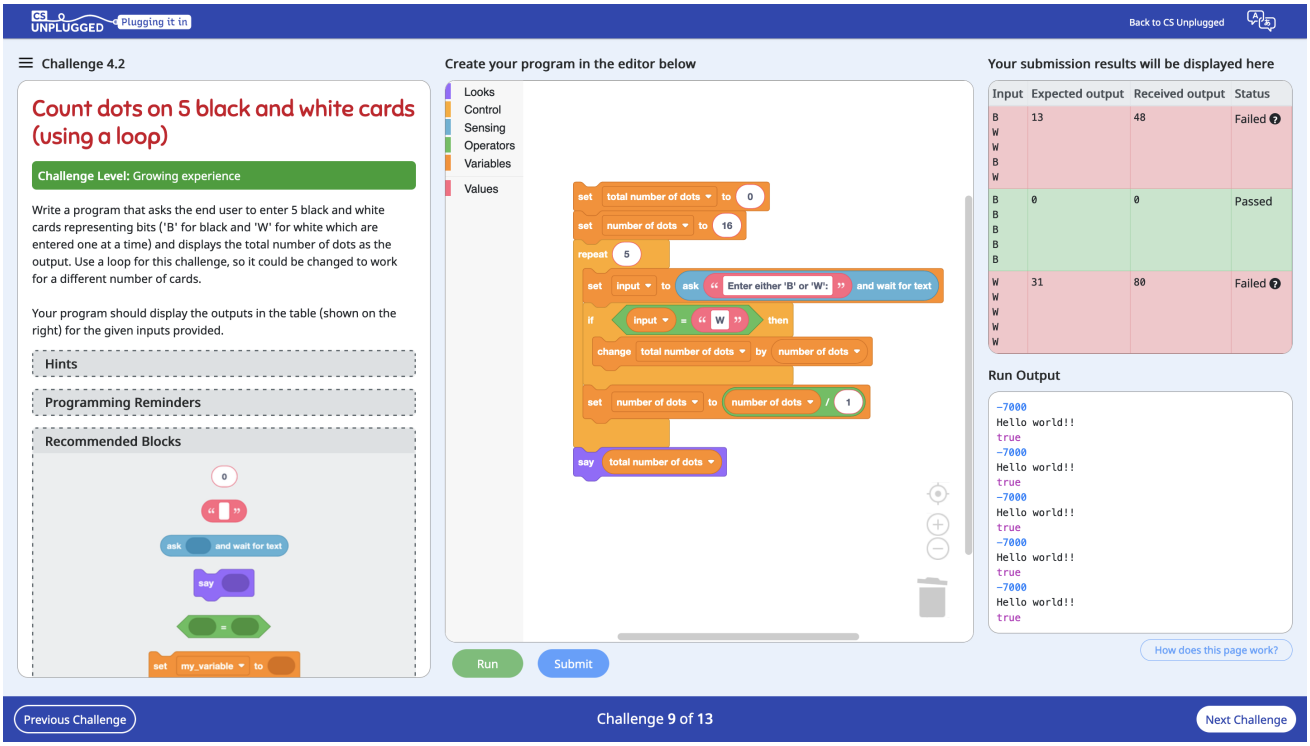

Fig. 20: Final version of the new block-based system.