NextBlocks: An Interactive Block Programming Platform

Duarte Pereira
Fernanda Barbosa
Carmen Morgado
dg.pereira@campus.fct.unl.pt
fb@fct.unl.pt
cpm@fct.unl.pt
Nova School of Science and Technology
Department of Computer Science
Caparica, Setúbal, Portugal

ABSTRACT

Since Seymour Papert's work with the Logo programming language in the 1960s, there has been a prevailing belief in the effectiveness of visual programming environments for teaching programming to children and novices. As these platforms evolved and became more prevalent, using block programming to teach children and beginners became increasingly more common. However, modern block programming platforms like Scratch and Code.org excessively restrict educators by not allowing them to create custom exercises in the platform. They also tend to make programming a solo activity, not allowing for collaboration and cooperation between learners (Scratch is an exception in this regard). Additionally, they are usually located in standalone websites instead of being implemented in locations that students already frequent regularly. Having identified these gaps in the field of block programming environments, this paper proposes NextBlocks, a new block programming platform implemented as a Moodle plugin. This platform enables educators to create custom exercises, emphasizing social perception and collaboration features. It supports features that are uncommon in block programming environments, contributing to a more interactive and engaging learning experience. Furthermore, being integrated into the Moodle Learning Management System makes NextBlocks more easily accessible within the educational framework. As an open-source platform, besides solving current challenges, it can also serve as a foundation for future expansion by the education community. This paper explores some of the unique features of NextBlocks, presents a case study on the platform, and discusses its potential contributions to enhancing programming education for beginners within a collaborative learning environment.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**; **Interactive learning environments**; **Learning management systems**; **Collaborative learning**; • **Social and professional topics** → **K-12 education**; **CS1**; • **Human-centered computing** → Collaborative and social computing systems and tools; *Graphical user interfaces.*

KEYWORDS

Visual Programming Environments, Block Programming, Moodle, Computer Science Education, Social Perception and Collaboration.

ACM Reference Format:
Duarte Pereira, Fernanda Barbosa, and Carmen Morgado. 2024. NextBlocks: An Interactive Block Programming Platform. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653609

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07
https://doi.org/10.1145/3649217.3653609

1 INTRODUCTION

Programming is an important skill to learn because it empowers individuals to create and innovate. Programming is not only a valuable skill, but also a means for developing computational thinking skills: the ability to formulate a problem to admit a computational solution [24]. According to Wing [23], computational thinking is the most beneficial skill that students can acquire from learning how to program. Block programming has been identified as a useful tool for learning text programming [16], and for developing computational thinking [25]. Block programming languages, as opposed to text-based programming languages, use graphical blocks representing programming concepts to compose programs, rather than text-based representations of those same concepts.

Currently, there is no widely used block programming environment that allows educators to create custom exercises. Such a feature would allow educators to adapt the exercises to the course content, and would thus make the platforms more versatile. Additionally, the existing block programming environments generally make programming a solo activity, by not having features that enable collaboration between learners.

By being integrated into the Moodle LMS as an activity, a block programming platform could benefit from being part of a centralized learning environment that students already frequent regularly. This would benefit students, who no longer would have to visit external sites to do their coursework, but also teachers, by giving them access to existing Moodle features, such as authentication, the gradebook, and coordination with other activities.

2 BACKGROUND AND RELATED WORK

This section will analyze and discuss concepts and learning tools that were relevant for the development of this project.

2.1 Block Programming

Block programming languages are a type of programming language that use visual blocks to represent programming constructs, as opposed to using a textual representation as in most traditional languages. Programs can be created by dragging, and arranging these blocks in a dedicated GUI. Block programming languages have been gaining popularity recently [5][19] for teaching introductory programming to children, beginners and non-programmers.

The history of block programming can be traced back to the work of MIT researcher Seymour Papert, who designed the **Logo** programming language in the 1960s [20]. Logo is a text-based programming language which uses a turtle graphics system to draw and control shapes and patterns on the computer screen [9]. Logo was the first programming language designed specifically to be used by children [20], and thus quickly gained popularity in schools and other educational settings.

In the 2000s, researchers at the MIT Media Lab started working on a new block programming language called **Scratch**, which released in 2007 [12]. This language was designed to be even more child-friendly than Logo [13], by representing programming concepts as visual blocks instead of text [14]. Scratch was also one of the first platforms to introduce a social element to programming, allowing users to share and edit each other's projects [18].

Scratch became popular in educational settings, and influenced subsequent block programming platforms and tools such as **Blockly** [8], **Snap!** [10], and **App Inventor** [1]. Block programming is currently used in a wide variety of settings, from elementary schools [26] to professional development environments [21].

Block programming languages have many advantages over traditional text-based programming languages: **(1)** They are more accessible and easier to learn, especially for non-programmers [22]; **(2)** They are more engaging, with colorful elements and interactive features; **(3)** With modern platforms there is no need to install language tools, as they run in the web browser [6]; **(4)** There is no need to memorize vocabulary, as all blocks are available in the interface [6]; **(5)** It reduces cognitive load by chunking complex constructs into blocks, reducing the number of information chunks that a learner needs to process [6]. For example, for a beginner, a Java for loop has many pieces of information (the keyword, initialization, condition, increment, *etc.*) while in an environment like Scratch, a loop has few pieces (the keyword and the upper bound) [4]; **(6)** It is less prone to errors, because block programming environments prevent users from composing programs with errors using the constrained direct manipulation mechanism, which only allows blocks to snap onto each other if they are compatible [6].

2.2 Blockly

Blockly[1] is an JavaScript library developed by Google that enables the creation of block-based programming languages and environments. It provides a block editor, featuring an area for selecting blocks and another for dragging and dropping blocks to compose programs, and a framework for generating text-based code in many languages, including JavaScript and Python [17].

Blockly is free and open-source software, licensed under the Apache License Version 2.0 [2]. The library has many useful features, like allowing users to export their code to text-based programming languages, or enabling the creation of custom blocks by the application developer. Blockly has been localized into over 40 languages, meaning it can be utilized all around the world to its fullest extent [2]. Blockly runs fully in the browser, meaning it is light on server resources. It supports basic programming constructs by default, like control flow (loops, functions, conditionals), variables, boolean and arithmetic operators, and more, and allows developers to implement custom blocks that perform more complex operations.

2.3 Moodle — Learning Management System

Moodle[3] [3] is a widely used Learning Management System (LMS) that has had a significant impact on education. Moodle was designed to provide educators with a platform for offering online courses, promoting cooperation, and involving students in an interactive learning environment.

2.3.1 Activities and Resources. In Moodle, educators have many options for creating and managing learning materials. In Moodle, these materials come in two different forms: **Resources** and **Activities**. A resource is a piece of content, like a portion of text or a video file, that is to be presented to the learner, who consumes it in a passive manner. Examples of resource types include the **File** and the **Book** resource. An activity, on the other hand, is something that students can engage with or contribute to directly, like **Survey**, **Quiz** and **Forum** activities.

2.3.2 Moodle Plugins. One of Moodle's main features is its modularity and extensibility, which it achieves with plugin support. Plugins are additional pieces of software that can be added to a Moodle site to extend and customize the it beyond its core functionalities. Administrators and educators can use them to customize Moodle to their requirements, by adding features, improving existing ones or changing the overall look and feel of the site.

There currently exist many Moodle plugins that enhance the learning experience. Plugins used in the context of programming education include:

- **CodeRunner**[4]: CodeRunner is a Moodle question type plugin that enables instructors to add programming exercises to quizzes, and supports various popular programming languages. Students can write and execute their code within the Moodle interface, where they also receive automated feedback on the execution of their code. CodeRunner features automated testing, which runs the submitted code and checks the output against the expected answer.
- **Virtual Programming Lab (VPL)**[5]: The VPL plugin adds a new activity type, and has similar functionalities to CodeRunner. It enables the development of programs inside of Moodle

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

[2] https://github.com/google/blockly/blob/develop/LICENSE

[3] https://moodle.com/

[4] https://coderunner.org.nz/

[5] https://vpl.dis.ulpgc.es

for submission and automated evaluation. What sets it apart from CodeRunner is the fact that it is a standalone activity type, not a question type to be used in quizzes. It also features a more advanced editor with syntax highlighting, a debugger, and features for preventing and detecting plagiarism.

Plugins that include social perception and collaboration features include:

- **Point of view**[6]: This plugin adds a block on the side of each piece of course content, featuring three *emoji*, each representing a specific reaction towards the material (easy, medium, hard). Students can click on one of the *emoji* to indicate their reaction towards the content. This plugin allows educators to gauge students' perception of the difficulty of the course content, helping identify potential areas for improvement and the topics where students might require additional assistance.
- **Annoto**[7]: The Annoto plugin adds a collaborative annotations system to any video posted on a Moodle page. It works as an overlay to any video, allowing students to post their comments or questions on a specific timestamp, which can be seen by any other students watching it. It also features a real-time chat system that allows any two users who are watching the video at the same time to communicate with each other. The functionality added by this plugin helps students stay engaged with the course content, by turning video watching, which is usually an individual activity, into a collaborative and interactive learning experience.

By creating a new Moodle activity type in the form of a plugin, we can combine the advantages of block programming with the potential reach of Moodle. All of the previously mentioned plugins influenced the development of NextBlocks. In particular, the Annoto plugin was a major inspiration, because it also transforms a typically solitary action (video watching) into a collaborative activity.

3 NEXTBLOCKS

The proposed platform, being a block programming environment, has much in common with other similar platforms in terms of the basic features. It features a *workspace*, where students can drag blocks in order to compose their programs. These blocks come from the *toolbox*, located on the left side of the interface. This area is divided into categories. The platform allows students to run their program and view the output inside the plugin interface. Like in other block programming environments, incompatible blocks cannot be composed. For example, it is not possible to place a multiplication expression inside the area for the condition of an if statement. The plugin interface is shown in Figure 1.

3.1 The Block Programming Environment

The NextBlocks block programming environment is implemented with the Blockly library. It is composed of a hierarchy of interface elements, each with its specific function. The main area, referred to from now on as the *workspace*, is where the student can drag blocks and snap them together in order to compose a program.

The *toolbox*, located to the right of the *workspace*, is the area from which blocks are selected. This area is divided into categories that group blocks together. For example, blocks related to math (numeric literals, arithmetic operators, *etc.*) are in one category, while blocks related to logic (the if block, boolean literals, boolean operators, *etc.*) are in another. In order to compose a program, the user clicks on one of these categories, chooses a block, and clicks and drags it onto the *workspace* where it can be placed or snapped onto other blocks.

The area to the right of the workspace is divided horizontally into two areas: the *Reactions* area and the *Tests* and *Output* area. The reactions area allows the students to communicate their feelings towards the exercise using one of 3 reaction emojis. This area will be further detailed in the following section.

The *Tests* and *Output* area is used for displaying information about code execution. This area features two buttons that are used to toggle between the *Output* and the *Tests* interfaces. The *Output* area displays the output of the code that is currently in the workspace. If there is a print statement in the code, the output is directed to this area. The *Tests* interface shows the results of the automated tests that are run on the code that is currently in the workspace. This information is presented as an accordion menu, where each header shows the number of the test and whether it the code passed or failed the test. If the user clicks on the header it expands, showing the inputs, the expected output, and the actual output of the test.

Under these interface elements, is a row of buttons related to the exercise. The *Save* button stores the current state of the workspace, allowing the user to save an in-progress version of his solution for restoring in the future without losing any data. The *Submit* button makes the exercise available for grading, and in case that the user has reached the submissions limit, it makes the workspace read-only. Additionally, the *Save* button also saves the workspace for future restoration, in the case that multiple submissions were enabled by the exercise creator. There is also a *Cancel* button, that discards all changes since the last save / submission, and redirects the user to the course's home page.

In the bottom is the *Chat* interface, that allows students and educators to discuss the exercise with each other.

3.2 Programming Features

In addition to common features found in other platforms, NextBlocks offers advanced functionality that set it apart from block programming platforms such as Scratch or Code.org.

- **Creation of exercises**: One of the main distinguishing features in the platform is the ability for educators to create their own programming exercises. It is possible to specify the exercise description and a set of automated tests, and the students can solve the exercise and submit their solution. This feature is not present in any other block programming environment that has been evaluated, and thus is something that should set this platform apart from the others. The exercise creation interface is shown in Figure 2.
- **Automatic code evaluation and feedback**: Manually grading programming assignments can be time consuming and is prone to errors, especially when an educator grades a large number of submissions. As a result, automated code grading

[6] https://moodle.org/plugins/block_point_view
[7] https://www.annoto.net/

Figure 1: The full NextBlocks interface, featuring a solution to the Uber fare calculation problem.

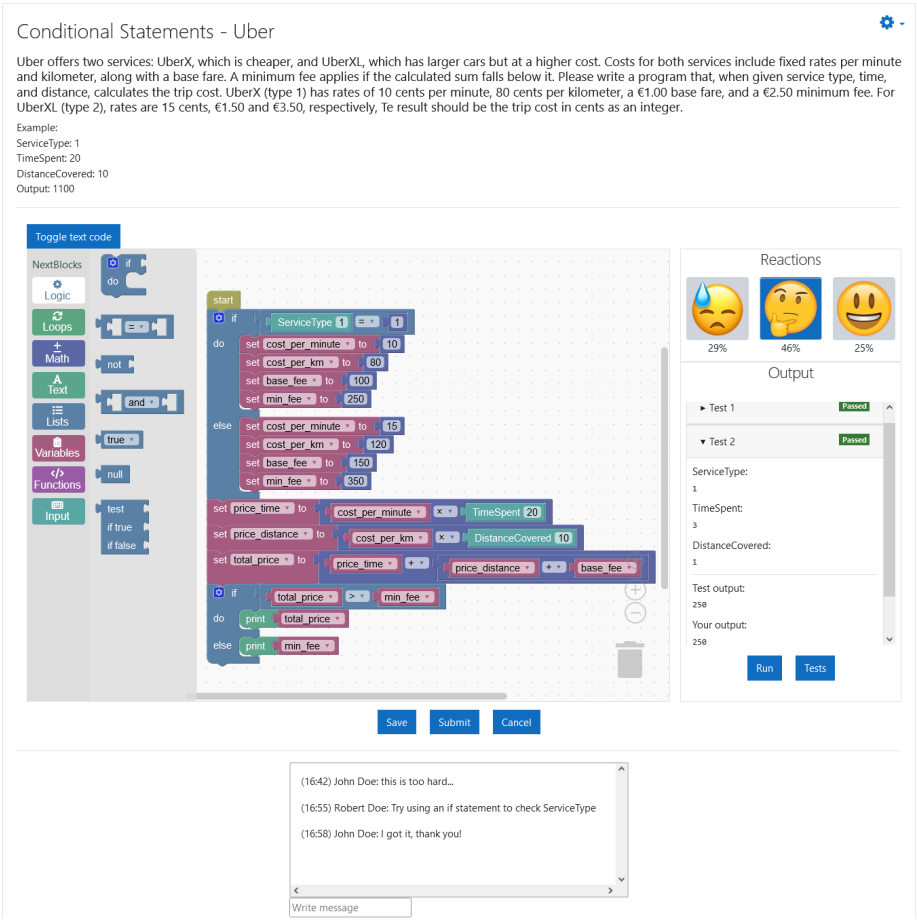
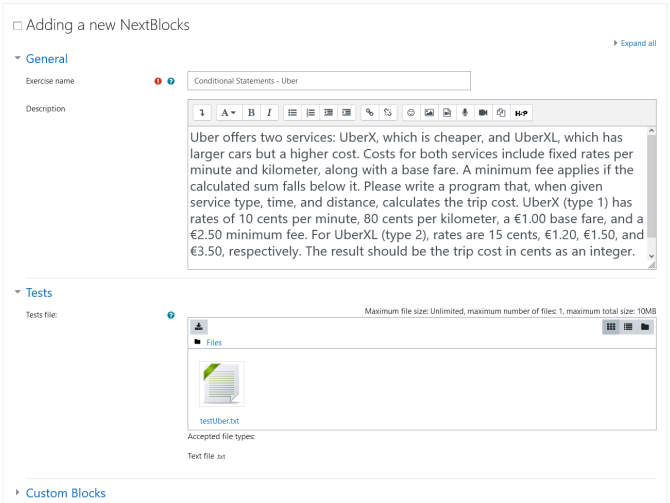


Figure 2: NextBlocks exercise creation interface



tools have become widely popular [15], because they can be used to grade large numbers of submissions in a short amount of time. As is common in Moodle programming plugins like CodeRunner and VPL, NextBlocks allows for educators to specify automated tests for the exercises that every submission has to pass in order to be marked as correct. Students can see information on the tests that passed and failed, which can help them in fixing the program.

- **Creation of custom blocks**: The platform allows educators to create their own custom blocks for a specific exercise for students to use while solving it. For example, if an exercise requires the use of a complex formula, but the goal is not for the students to implement the formula, the educator can define a block for that formula. Consequently, students will be able to skip that specific step and focus on the relevant part of the exercise. The custom blocks are added in the exercise creation interface.
- **Restriction of programming primitives**: A feature that some block programming platforms implement is the restriction of the programming primitives allowed on a per-exercise level. For example, if an exercise is meant to teach the if statement, and loops have not been taught yet, the loop blocks

will not show up in the programming interface if the educator sets up the exercise to do so. Another case would be restricting the number of times that certain blocks can be used. For example, if the exercise is meant to teach loops, it does not allow repetition of other blocks. This feature is in development, and will allow the creator of each exercise to specify the number of times that each block can be used.

3.3 Social Perception and Collaboration Features

The social perception and collaboration features are some of the aspects that differentiate NextBlocks from existing block programming platforms, and for that reason, they should be discussed.

- **Reactions**: Inspired by the **Point of View** Moodle plugin, this feature allows students to communicate their attitude towards the programming exercises. If a student is struggling with an exercise, they can mark it as difficult. This feature allows educators to gauge their students' difficulties, or lack thereof, with the material, helping them identify areas of potential improvement. This interface is shown in Figure 1.
- **Comments**: This feature allows students and educators to leave notes on submitted exercises, related to specific sections of the code. The commenting system allows students to leave annotations on the code, for example, asking if the code is well written, or for educators to leave comments, suggesting students a different way to solve the problem, or giving them tips if they did not pass the automated tests. These comments can be added both before or after submission, allowing students to view comments added by educators during the grading process.
- **Real time chat**: It has been shown that integrating instant-messaging systems in educational contexts and tools can contribute to improve the learning experience of students [11], so for this reason, it was decided to add a chat system to NextBlocks. Moodle features messaging systems, but these are either for conversations between any two Moodle users that use a Moodle site, or for chats between two users in a course. Having chat functionality integrated in the activity gives students the ability to communicate about specific exercises, ask each other or educators for help, or give advice. This feature is inspired by the **Annoto** plugin, which enables students to chat about videos. This feature would encourage students to engage more with the exercises and learn from each other, leading to a better learning experience.

4 EVALUATION

A small round of user tests was conducted, to evaluate NextBlocks' usability and potential usefulness in the classroom.

The student-facing interface was evaluated with a group of 8 students, who used the platform by following a script similar to the case study, ensuring that they interacted with its various features. The System Usability Scale (SUS) [7] was used for evaluating the platform's usability. The average result of the questionnaire was 77.5, meaning that students were overall satisfied with the platform's usability. When asked about their satisfaction with specific features, responses were more mixed. Running tests, automated grading and reactions were highly rated, but chat and comments had a lower rating. The issues with these two features had already been identified, and are currently being fixed.

As for students' perception of the impact of NextBlocks' features on the learning experience, results were very positive for the tests, automated grading, and commenting features. However, chat and reactions received relatively lower (but still positive) ratings, which was not expected. Further user testing is needed to better understand user perspectives on these features. When asked about their overall satisfaction with NextBlocks, 87.5% of responses were positive, which is consistent with the usability score.

As for the educator's side of the interface, a round of user tests with 6 university-level educators was conducted. They were tasked with creating an exercise, and then viewing a submission, grading it, and adding a comment, among other tasks.

The average result for the educators' SUS questionnaire was 78.3, suggesting that educators had a favorable opinion on the platform's usability. When rating specific features' usability, all were rated positively (average rating of 4.1 out of 5), but chat was rated lower than the others (3.6 out of 5). As for the features' potential impact, educators rated the tests and post-submission comments very highly, but the other features also had positive results. When asked about their overall satisfaction with the platform, 83.3% of respondents answered positively, reflecting educator's satisfaction with the platform.

In conclusion, reception to the NextBlocks platform was positive, both for the student-facing and educator-facing interfaces. Users showed interest in the platform and rated its usability positively.

5 CASE STUDY

This case study examines some of NextBlocks' functionalities, a block programming platform implemented as a Moodle plugin. NextBlocks addresses limitations in existing platforms by focusing on social perception and collaboration features, and enabling educators to adapt the exercises to their own curriculum. This study uses a hypothetical use case to analyze how NextBlocks could contribute to an interactive and engaging learning experience for beginners.

5.1 Objectives

The primary objectives of this case study are as follows:

- **Evaluate customizability**: Assess the extent to which NextBlocks addresses the limitation of customizability observed in existing block programming platforms, allowing educators to create custom exercises that fit their needs.
- **Explore social learning**: Examine NextBlock's collaboration and perception features within to determine how they facilitate cooperative learning experiences among beginners.
- **Illustrate Moodle integration**: Convey how the proposed platform is integrated into Moodle as a plugin.

5.2 Scenario

The theoretical scenario that will be presented in this section will be used to present a possible usage of Nextblocks, by presenting the perspectives of the two main user classes of the NextBlocks plugin: the educator and the student.

5.2.1 Educator's Perspective: Dr. Robert Doe is a university professor teaching an introductory Computer Science class. He notices students are having trouble understanding the conditional statements, so he decides to try integrating NextBlocks into his class by designing a custom exercise.

Having formulated an exercise that would require a correct application of the conditional statement, Dr. Doe goes to Moodle to create the NextBlocks activity. He goes onto the course page, clicks on *Add activity or resource*, and chooses NextBlocks from the list. In the NextBlocks activity creation form, he inputs the name and description of the activity. He then looks at all of the available options, noticing the *Tests* section. He opens the tab, revealing a file selector and a help icon. He clicks this icon, revealing an info box explaining that the tests can be used to let students validate their code on their machines before submission, and to automatically evaluate students' submissions. He reads the specification for the file format, and decides to create one with a few test cases.

After this he looks at the other options, deciding to only open the *Submissions* tab to allow for unlimited submissions, and to open the *Grade* tab in order to set the grade scale to point grading. He then clicks the *Save and return to course* button, making the exercise available to the students.

5.2.2 Student's Perspective: Mary Sue is a first-year student taking Prof. Doe's class, but she's having difficulty understanding if statements. While studying, she noticed a new exercise on the course's Moodle page and decided to open it.

Upon loading the page, she is faced with an unfamiliar interface. She clicks one of the categories on the left side of the screen, revealing a list of blocks representing programming concepts. She selects the print and input blocks, drags them to the workspace, snaps them together, and types "hello" in the input field. After clicking the *Run* button, she could see the program output. Having figured out the basics of this exercise type, she tried to solve the exercise.

After reading the exercise description, she notices that there are 3 blocks in the workspace called ServiceType, TimeSpent and DistanceCovered that cannot be removed. She assumes that those blocks represent the inputs mentioned in the exercise description, so she uses them to build her solution.

Mary successfully builds a solution for the UberX option, but she could not figure out how to check whether to calculate the fare for UberX or UberXL. She tests her solution by clicking the *Tests* button, but only the tests that use the UberX option pass, the others fail. Not being able to progress, she uses the *chat* to ask her classmates. A few moments later a classmate responds, recommending that she try using the ServiceType input inside an if statement. In that moment, Mary realizes that she can use the if statement to check if ServiceType is 1 or 2, indicating whether the service is UberX or UberXL. Mary thanks her classmate, updates her solution and runs the tests, passing all of them. She considers the exercise to be relatively difficult, so she reacts to the exercise with the thinking face *emoji* and submits her solution.

5.2.3 After Submission: After the submissions deadline for the exercise, Prof. Doe goes to the Moodle gradebook, and sees that the students did well in the exercise, achieving decent scores in the automatic grader. However, the most common reaction to the exercise was the thinking face *emoji*, meaning that students considered the exercise to be somewhat difficult. He opens Mary's solution, seeing that she did well, leaves some comments with suggestions on how to improve her code, and moves on to the next student.

5.3 Analysis and Discussion

The previous scenario shows how NextBlocks could positively impact the learning experience, by enabling educators to create custom exercises fitting their curriculum and providing social perception and collaboration features that help students and educators feel more connected during the learning process. By reducing cognitive load and removing the need to memorize vocabulary, block programming can help students advance their programming skills, allowing them to focus on the reasoning process and not worry about the more formal aspects of programming.

5.3.1 Customizability. One of the primary objectives of this case study was to evaluate the extent to which NextBlocks addresses the limitations of customizability observed in existing block programming platforms. By supporting the creation of custom exercises, NextBlocks allowed Prof. Doe to adapt the coursework to the course content, and not the other way around. This demonstrates NextBlocks' potential in providing a more flexible and adaptive learning environment.

5.3.2 Social Perception and Collaboration. Another goal for this case study was to explore the social features present in NextBlocks. As seen from Mary Sue's experience, NextBlocks can facilitate cooperative learning experiences among learners. Mary was able to seek help from classmates using the chat feature, and Prof. Doe was able to see that although students considered the exercise to be relatively difficult, they were mostly able to solve it correctly.

6 CONCLUSION

Block programming has been shown to have advantages over traditional programming environments when used to teach programming to beginners. Currently, block programming platforms are generally in their own websites, so by combining the advantages of block programming with the accessibility of an LMS like Moodle, NextBlocks has the potential to enhance the learning experience for both students and educators. It addresses the limitations of existing block programming platforms through customizability and a focus on social learning, enabling educators to create exercises tailored to their curriculum.

As technology continues to play a crucial role in education, platforms like NextBlocks can contribute significantly to the evolution of programming education. The platform's focus on enhancing engagement and fostering collaboration align with Moodle's goals of providing a user-friendly learning experience. This integration not only benefits from Moodle's existing features but also extends the capabilities of the learning management system to offer a more interactive programming education environment.

ACKNOWLEDGMENTS

This work is supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP

REFERENCES

[1] 2010. About App Inventor. https://web.archive.org/web/20100811023417/http://appinventor.googlelabs.com/about/. Accessed: 04/07/2023.

[2] 2022. What is Blockly. https://developers.google.com/blockly/guides/overview Accessed: 12/07/2023.

[3] 2023. Documentation. https://docs.moodle.org/403/en/Main_page Accessed: 21/01/2024.

[4] 2023. Repeat () (block). https://en.scratch-wiki.info/wiki/Repeat_()_(block). Accessed: 04/07/2023.

[5] David Bau, D. Anthony Bau, Mathew Dawson, and C. Sydney Pickens. 2015. Pencil Code: Block Code for a Text World. In *Proceedings of the 14th International Conference on Interaction Design and Children* (Boston, Massachusetts) *(IDC '15)*. Association for Computing Machinery, New York, NY, USA, 445–448. https://doi.org/10.1145/2771839.2771875

[6] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable Programming: Blocks and Beyond. *Commun. ACM* 60, 6 (may 2017), 72–80. https://doi.org/10.1145/3015455

[7] John Brooke. 1995. SUS: A quick and dirty usability scale. *Usability Eval. Ind.* 189 (11 1995).

[8] Neil Fraser. 2015. Ten things we've learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. 49–50. https://doi.org/10.1109/BLOCKS.2015.7369000

[9] Ronald Goldman, Scott Schaefer, and Tao Ju. 2004. Turtle Geometry in Computer Graphics and Computer Aided Design. *Computer-Aided Design* 36 (06 2004). https://doi.org/10.1016/j.cad.2003.10.005

[10] Brian Harvey and Jens Mönig. 2010. Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists? *Constructionism* 2010 (01 2010).

[11] Carmen Pérez-Fragoso Juan Contreras-Castillo and Jesus Favela. 2006. Assessing the use of instant messaging in online learning environments. *Interactive Learning Environments* 14, 3 (2006), 205–218. https://doi.org/10.1080/10494820600853876 arXiv:https://doi.org/10.1080/10494820600853876

[12] Yasmin B. Kafai. 2003. A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities.

[13] David J. Malan and Henry H. Leitner. 2007. Scratch for Budding Computer Scientists. *SIGCSE Bull.* 39, 1 (mar 2007), 223–227. https://doi.org/10.1145/1227504.1227388

[14] John Maloney, Leo Burd, Yasmin Kafai, Natalie Rusk, Brian Silverman, and Mitchel Resnick. 2004. Scratch: A Sneak Preview. 104–109. https://doi.org/10.1109/C5.2004.33

[15] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2024. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Trans. Comput. Educ.* 24, 1, Article 10 (feb 2024), 43 pages. https://doi.org/10.1145/3636515

[16] Barbara Moskal, Deborah Lurie, and Stephen Cooper. 2004. Evaluating the Effectiveness of a New Instructional Approach. *SIGCSE Bull.* 36, 1 (mar 2004), 75–79. https://doi.org/10.1145/1028174.971328

[17] Erik Pasternak, Rachel Fenichel, and Andrew N. Marshall. 2017. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*. 21–24. https://doi.org/10.1109/BLOCKS.2017.8120404

[18] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (nov 2009), 60–67. https://doi.org/10.1145/1592761.1592779

[19] Mitchel Resnick and Natalie Rusk. 2020. Coding at a Crossroads. *Commun. ACM* 63, 11 (oct 2020), 120–127. https://doi.org/10.1145/3375546

[20] Cynthia Solomon, Brian Harvey, Ken Kahn, Henry Lieberman, Mark L. Miller, Margaret Minsky, Artemis Papert, and Brian Silverman. 2020. History of Logo. *Proc. ACM Program. Lang.* 4, HOPL, Article 79 (jun 2020), 66 pages. https://doi.org/10.1145/3386329

[21] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018).

[22] David Weintrop and Uri Wilensky. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* 18, 1, Article 3 (oct 2017), 25 pages. https://doi.org/10.1145/3089799

[23] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (mar 2006), 33–35. https://doi.org/10.1145/1118178.1118215

[24] Jeannette M. Wing. 2010. Computational Thinking: What and Why? https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

[25] LeChen Zhang, Jalal Nouri, and Lennart Rolandsson. 2020. Progression Of Computational Thinking Skills In Swedish Compulsory Schools With Block-Based Programming. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE'20)*. Association for Computing Machinery, New York, NY, USA, 66–75. https://doi.org/10.1145/3373165.3373173

[26] Nikolaos C. Zygouris, Aikaterini Striftou, Antonios N. Dadaliaris, George I. Stamoulis, Apostolos C. Xenakis, and Denis Vavougios. 2017. The use of LEGO mindstorms in elementary schools. In *2017 IEEE Global Engineering Education Conference (EDUCON)*. 514–516. https://doi.org/10.1109/EDUCON.2017.7942895