International Journal on Advanced Science Engineering Information Technology

Vol.12 (2022) No. 5
ISSN: 2088-5334

Adoption of Visual Programming Environments in Programming Learning

Qais Batiha [a,*], Noraidah Sahari [a], Nazatul Aini [a], Noorazean Mohd [a]

[a] *Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi, 3600, Malaysia*
Corresponding author: [*]P93243@siswa.ukm.edu.my

***Abstract*— Programming education is gradually integrated into the school and university curricula. Accordingly, studies in the Computer Science education field have highlighted issues such as high failure rate, memorizing, bugs, the complexity of concepts, motivation, and uconfidence faced by students when learning a programming language, specifically object-oriented programming. These issues require specific learning environments to reach the target audience. Therefore, the objectives of this article are to identify the issues based on previous work and to verify those issues by interview feedback conducted with lecturers in the Department of Computer Science and Information Technology at the Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia. Following that, the theoretical principles underpinning environments were studied to explore the suitability of these environments for university education based on the identified issues. The investigated environments included Turtle Graphics, Alice, BlueJ, Greenfoot, Snap!, and NetsBlox, a co-located collaborative block-based programming, and OOPP were studied and used to teach and learn object-oriented programming. Based on the interviews with experts, we found that students still had issues when learning programming, which involved memorizing, bugs, the complexity of concepts, unconfidence, communicating with students during a calamity (distance learning), and a number of students in labs. We conclude that these environments focus on some issues and ignore others, and no single environment satisfies all these issues, which causes students to be demotivated. In a further study, all these issues will be addressed by developing a learning environment, and its effectiveness shall be tested.**

***Keywords*—Block-based programming; collaborative learning; object-oriented programming (OOP); syntax error; visual programming languages (VPLs).**

Manuscript received 11 Jun. 2021; revised 14 Apr. 2022; accepted 13 May 2022. Date of publication 31 Oct. 2022.

I. INTRODUCTION

Programming has been seen as an obligatory course in secondary schools and the first year of Computer Science degrees for the past 20 years. Therefore, many challenges should be addressed, such as the high failure rate in programming courses that has been a subject of extensive scrutiny [1]–[4]. This situation indicates the incompatibility of traditional learning methods, including the programming for novice students, which is possibly due to the difficulty of object-oriented concepts, demotivation, and students' lack of confidence in the inadequately developed novice solutions. Similarly, the Covid-19 crisis spreading worldwide at the end of 2019 has greatly affected education and hampered the educational process. Subsequently, students and lecturers are isolated, while the students' access to new knowledge from lecturers has become challenging [5].

Learning programming involves computing principles, which include memorizing, algorithms, logic, patterns, decomposition, abstraction, computational thinking, and evaluation. Hence, students' problems in this programming could be resolved using these principles [6]. It was argued in some studies [4], [7] that first-year students generally had insufficient time, no programming experience, or were unfamiliar with programming concepts. Besides, the requirements [6] for developing better programming skills (e.g., object, inheritance, selection, and repetition) in individuals and the production of future citizens with sufficient skills have been inadequate [8]. Additionally, several issues have emerged in the teaching and learning programming courses for students and instructors. Programming concepts such as syntax error, language syntax, code quality, and code tracking may impede the students from effectively learning a programming language and maintaining their motivation. Programming courses normally recorded a

high drop-out rate, implying the considerable challenges faced by students in learning programming [9], [10].

In a typical traditional programming class, students are encouraged to participate in learning activities through program development using specific languages. This phenomenon normally occurs after introducing a related textbook [11]. However, it should be emphasized that learning programming goes beyond reading educational books and understanding programming concepts [12]. Students must also master programming concepts and computational thinking, the primary foundations of developing an individual's programming skills. Additionally, customary programming courses solely provide text programming, given that students today are more acclimatized to computer-based (graphical) environments such as visualization tools, program development, or debugging. Thus, textual programming and command lines would appear foreign and unattractive [13].

Visual Programming Languages (VPLs) encompass languages based on graphical elements where the text is eliminated or used only to some extent. For instance, the first VPL encompasses the graphical system, Seymour Papert, which was introduced in the 1960s [14]. This was followed by the VPLs' introduction of various iterations and variants, as shown through the puzzle blocks, including Google Blockly [15]. These VPLs are not text-based but employ drag-and-drop and other spatial actions. For the programmers, VPLs increase programming availability for a specific audience.

The use of VPLs enhances the precision of the execution of programming tasks and enables faster execution of programming tasks [16]. Besides, it contributes to more opportunities for programmers to write programs based on four common strategies embraced by VPLs: concreteness, directness, explicitness, and immediate visual feedback [17]. To develop an interactive environment capable of enhancing student learning, the issues faced by students in learning object-oriented programming (OOP) must be identified, and the methods of improving learning programming must be determined. This research addressed the expert's perspectives on the issues of learning OOP. Based on the determined issues, this research offered a comparative analysis of the freely available teaching/learning programming environments. The study results then presented the features important to the environment to enhance students' motivation.

Based on previous studies, this study approached the issues students face in learning programming. Following that, open-ended questions were developed to verify these issues from the perspective of experts. The study also focused on several freely available environments that focused on learning OOP. The theoretical principles underlying these environments were examined to qualitatively assess the environments and explore their suitability within the setting of secondary and university based on the problems identified in previous studies and interviews with experts. This paper attempts to answer the following research question: RQ1) What are the experts' perspectives on the issues (concept difficulty, memorizing, bugs, number of students, and unconfident) leading to the demotivation of OOP learners?, and RQ2) Did the seven object-oriented learning programming environments tackle motivation issues?

A. Literature review

Many works of research have been conducted to study issues faced by students in learning OOP. Alternatives to teaching methods about OOP topics were determined as OOP is a complex and compulsory subject for students. According to Hnin and Zaw [18], students are faced with issues in memorizing reserved words and syntax errors in writing code. Additionally, the slow pace of problem-solving and production of lower-quality software leads to less confidence in learning programming [19]. During the Coronavirus pandemic, students' difficulty in communicating with teachers during curfews and distance learning leads to slow learning progress and frustration [20], [21].

The problems faced by students in programming courses have been addressed in numerous studies, such as the success/failure rates, the complexity of the programming material, syntax error, error messages, individual learning (solo programming), and the number of students per lab [2]–[4], [10], [22], [23]. Nonetheless, efficient and innovative methods for learning basic programming skills are present. These methods motivate students to learn while being valuable additions to the teaching methodology. Many environments for graphical programming with high educational value are available and more attractive compared to textual programming. Some of these environments have been suggested as elements of school learning methodology for problem-solving development [24].

For novice learners, this entire procedure may appear extremely advanced to integrate. The most important part of the process is logic, which is the initial step in developing a comprehensive program. Notably, given that many high-level languages (e.g., Java and C++) contain comparable semantic and syntax rules, writing statements using a particular programming language requires sufficient knowledge of the programmer regarding language structure, the solution to coding errors, and the ability to memorize certain concepts. Hence, the complex structure of programming languages has value to professionals, although it is not a pedagogical value. Consequently, many learners are faced with difficulties in understanding the fundamental concepts, including those related to data structure management or the generation of an algorithm to resolve an issue. Besides, teachers further emphasize the difficulties in programming laboratory sessions due to various factors, including large numbers of students per laboratory and communicating with students during a calamity such as Covid-19.

Visual programming language solves issues novices face through a focus on the programming language logic while removing the need to learn the programming language syntax beforehand or write many code lines [24]. These tools allow novices to master programmatic concepts before understanding the goal language's syntax and focusing on problem logic before syntax. Notably, VPLs are an innovative and increasingly popular solution for programming novices in the classroom due to their high enjoyability and motivation [25], [26].

Tsai [27] studied using VPL (block-based) to teach students about problem-solving and found a significant improvement. Meanwhile, the second research test explored the association between fun and programming, which recorded that enjoyability was a good motivator to practice

and program new things. By introducing students to programming, VPL was an excellent motivator for novices who wished to learn to program, as recorded across several programming environments [28], [29].

A comparative study recorded that students obtained higher scores in concept assessments at the high school level after five weeks of studying using VPL (block-based) compared to similar text-based alternatives [30]. Notably, the VPL to programming successfully engaged novice programmers from historically underrepresented groups [31], [32]. Also, Weintrop [30] demonstrated that students who were learning using a VPL curriculum were properly prepared for future learning using a text-based approach.

Failure to adapt to diverse learning styles leads to unsuccessful traditional learning programming methodology, followed by a lack of motivation with the traditional teacher-centered pedagogy approach. In contrast, the notion of constructivism assumes that student activities and social interactions amongst peers (e.g., virtual pair programming [33], [34] and discussion forums [35]) have the same effect on learning and knowledge.

Virtual pair programming is a form of collaborative learning where the programmers are located in separate places without the need for in-class face-to-face interaction via synchronous or asynchronous interactions (video call, voice call, or shared desktop) [36]. Computer programming courses may be challenging, besides the significant falling grades among students due to the lack of time and motivation. According to Adeliyi et al. [19], pair programming improved programming and computer practice learning. Several studies performed empirical research on pair programming effects in the computer field. Subsequently, Adeliyi et al. [19] and Hughes et al. [37] indicated that pair programming led to seven positive outcomes, including mutual encouragement, reciprocal supervision, correction, re-examination, mutual trust, self-confidence, and sharing of expertise.

The collaboration technology could improve students' programming, minimize programming errors and design efficiency, enhance students' happiness and skills, team cooperation morale, self-confidence, and students' learning activities. Besides, social interactions lead to the students' reliance on sources other than the teachers instead of perceiving the teachers as the only source for acquiring skills and seeking suggestions, which reduces the tutors' work burden. Hsu et al. [38] and Tsompanoudi et al. [39] found that encouraging students to pair in a programming course was more effective for students engaging in high-level languages, leading to a higher percentage of students completing the course. Similarly, the distinctions between the learners who work "solo" and those working in pairs were made in a 2018 paper, contributing to the conclusion that programming knowledge, collaboration, and technical skills were further improved with a partnership [40]. According to Cheng and Lei [41] and Hughes et al. [37], discussions between students allow them to gain more insights, play a role in communication with others, and increase their confidence. Additionally, this phenomenon reduces communication problems between students and lecturers while increasing students' achievements and the lecturer's adequate supervision.

To address the issues stated in previous studies related to learning OOP, the features of a programming environment could be identified to enhance students' learning. According to Xinogalos et al. [22], the main features of a programming environment that would improve the motivation to learn OOP are user-friendliness with a simple design, visualization of OOP concepts, easy understanding of error messages with a recommendation to solve the issue, puzzle-like statements (editor), and execution. Meanwhile, de Oliveira et al. [40], and Brown and Wilson [12] suggested that pair programming as a "student-student" was the most powerful feature in increasing students' confidence and code solution quality. Laurel [42] suggested that learning programming should include group interaction (e.g., forums), modularity (e.g., variety of a puzzle piece), and expressiveness, which would motivate students to a complex programming process and enhance communication.

II. Materials and Method

An interview with five experts is sufficient to discover the presence of an issue. Having more than five participants would have the analysis too complicated and would have generated an excessive amount of data [43]. Three interviewees were male, while the other two were female and were selected based on their reputation and experience as lecturers for several years (ranging from 5 to 18 years) in Computer Science and Information Technology faculty. Each specialist published on programming challenges and had tough bachelor's programming courses. A total of 30 publications were published by three of the participants. The interviews were conducted with individuals who strongly relate to learning OOP. They were invited to share their perspective on the questions obtained from previous work. The interview started with an explanation about the purpose of the research and the reason for using the interview method. The interviews were conducted at the Faculty of Information Science and Technology, UKM, each lasting for a maximum of one to two hours.

Following that, we present an assessment of seven educational learning environments that were criticized based on literature review and issues raised by experts. In this context, relevant literature has been studied and carried out an assignment in each one of the following environments: the well-known IDE "objects-first" by BlueJ [44]; OOPP [45] hybrid environment; Greenfoot [44] learning and microworld environment; Turtle graphics [14] physically oriented environment; Snap! [46] game-oriented environment, and the collaboration extension environment of NetsBlox [47]; Alice [48] 3D storytelling and animation environment, and AliCe-ViLlagE [49] virtual pair programming environment; Multi-Device Grace [50] touch-screen block-based programming. A comparative analysis of seven educational programming environments was performed. Specific educational programming environments were selected from a wide range of environments based on four main criteria. This research used interactive programming environments that focused on OOP learning, incorporated a wide variety of features to assist novices in various methods, implemented free availability, and influenced student practice. A description of each environment technological and historical created a context for this study.

Seymour Papert introduced Turtle graphics in the late 1960s as parts of the Logo programming language [14]. The

project was logo-based and created to support Papert's notion of mathematical education. Turtle graphics can be applied in many different languages and are commonly used in basic programming education concepts for novices. The environment of the turtle graphics was fashioned from the MIT project of a robot that resembled a turtle, while its library encompassed the concept of a "turtle", which could move across a 2D plane. A pen driving with the turtle could be positioned on or off the ground, which led to the tracking of the turtle's movement (see Fig. 1). The turtle graphics design was inspired by a physical and graphical model, which was similar to the act of drawing on paper with a pen. This concept would be simple to grasp for students.

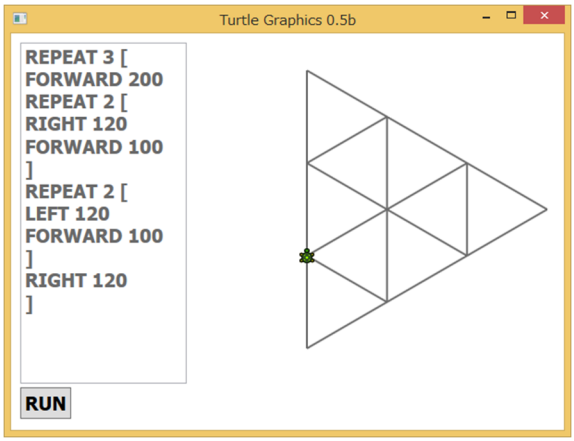

Fig. 1 Turtle Graphics environment

Alice is a block-based storytelling approach that depends on the objects-first approach. Alice was introduced to female students in middle school as a computer program grounded on interactive 3D animated stories [51]. The program was introduced in 2005 by a group of researchers at Carnegie Mellon University, led by Randy Pausch when American students' interest in Computer Science dropped by 50%–70% [48]. Therefore, Alice was introduced as an environment to improve the ability of at-risk undergraduate computing students to achieve success in Computer Science 1 and beyond [52]. In 2014, Alice had an extension called AliCe-ViLlagE to enhance students' confidence and communication skills by adding the pair programming feature to the Alice environment [49]. Alice and AliCe-ViLlagE allowed students to observe the progress of animated programs promptly. Furthermore, the outstanding visual feedback offered by Alice allowed students to relate the program "piece" to the action they observed in animations (see Fig. 2).

Fig. 2 Alice environment

Besides, Alice provided many features presented as Actions (two categories: an object to execute motion and transform the physical nature of an object), functions, titled instructions, decisions, recursion/looping, and events/ interactions [48].

Initially introduced in 1999, BlueJ entails the re-implementation of the Blue environment (applied using a C++ programming language) for Java language [44]. BlueJ was particularly created to introduce the teaching of a programming language in a Computer Science course at Sydney University using a 2D environment platform. It also enables the identification of classes and the relationship between them. In this case, the UML-like notation is used (refer to Fig. 3B). When the classes are compiled (see Fig. 3A), students could interactively instantiate objects (refer to Fig. 3D) to obtain a simple representation of objects created on the bench objects (refer to Fig. 3C). Accordingly, these objects could be checked, while their methods could be executed. The main advantage of BlueJ is the apparent separation of the concepts of objects and classes, which could be inspected and interacted with [44].

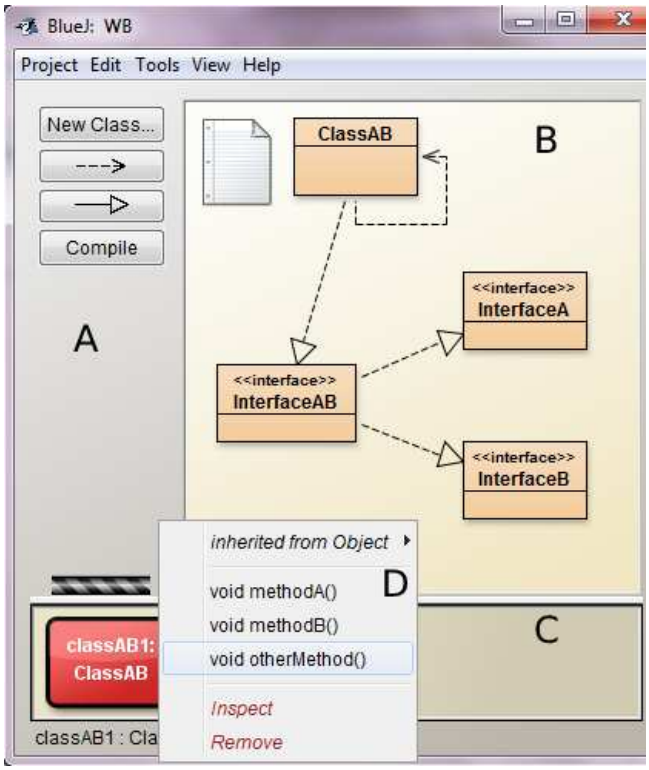

Fig. 3 BlueJ environment

The Greenfoot environment is an education-incorporated development for teaching and learning programming among novice learners [44]. Its operation in a 2D environment allows the development of interactive projects. Greenfoot blends programming in Java with graphical and interactive outputs as the standards and a text-based OOP language. This environment was designed for users aged 14 years and above, including users in college and university. The Greenfoot interface was designed to encourage novices to modify the behavior and visualization of objects, which could be performed through the scripting of new functions or the alteration of Java code from the editor.

As observed in Fig. 4, the GUI in Greenfoot consists of a staging area, which primarily presents the Greenfoot world with one actor at the minimum (see Fig. 4A), while the stage area is where the program is executed. The second section is located on the right (see Fig. 4B) based on BlueJ and presents the tree of all the available classes and their relationship with inheritance. These elements could be edited by clicking on the respective tabs. Additionally, World and Actor (Object) encompass the two visible upper classes of the system, which are neither modifiable nor removable. Java compiler and an integrated debugger are included in Greenfoot [44]. Notably, the structural syntax error could be a challenge to individuals who are still unfamiliar with the environment and faced with issues in diagnosing and fixing it.

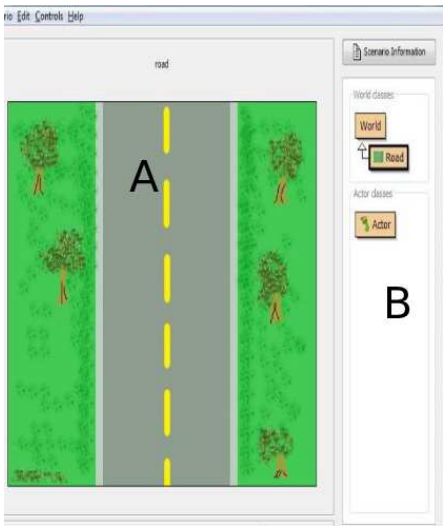

Fig. 4 Greenfoot environment

A co-located collaborative block-based programming, Multi-Device Grace, was constructed to explore Block-programming on all touch-screen devices developed in 2019 by Selwyn-Smith, et al. [50], which also utilized a 2D environment platform. Multi-Device Grace comprises four sections (see Fig. 5), specifically the toolbox (see Fig. 5B) that comprises the programming commands. The workspace (see Fig. 5A) is the place where students could execute these dragged commands by clicking on Run commands. The system could further create the target code by clicking on the code view command from the command menu (see Fig. 5D). Fig. 5C presents the frame of the application, which is identified in the output area.

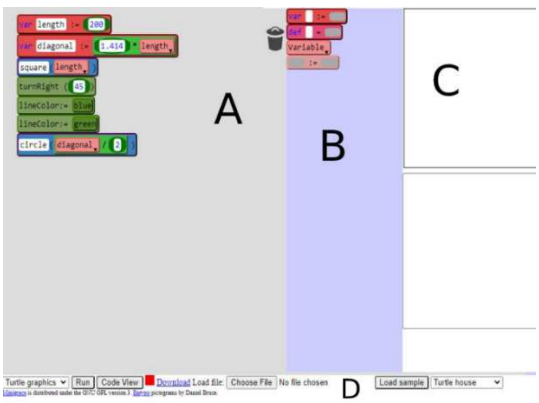

Fig. 5 Multi-Device Grace

Known in the early versions as BYOB (Build Your Own Blocks), Snap! is a Block-based programming language that also includes natural objects, specifically the sprites. It allows students to define new blocks to expand the language. The power of language focuses on children, high schools, and university students. Given that Snap! is implemented in JavaScript and operates within a browser, it does not require local installation procedures. As an extension of Snap!, NetsBlox (see Fig. 6) is a visual programming paradigm, with the environment providing networking features that enable students to create distributed applications to enhance the students' confidence. According to Google Docs collaboration style, the environment allows students to work together from different computers on the same project.

The development of OOPP was as the support for OOP, which was introduced by Alberto Ferrari in 2017 [45]. It is also a part of Google Blockly as an open-source developer library that allows the addition of block-based coding into an application [15]. This environment is specifically targeted to facilitate learning Java programming courses for both school students and the university.

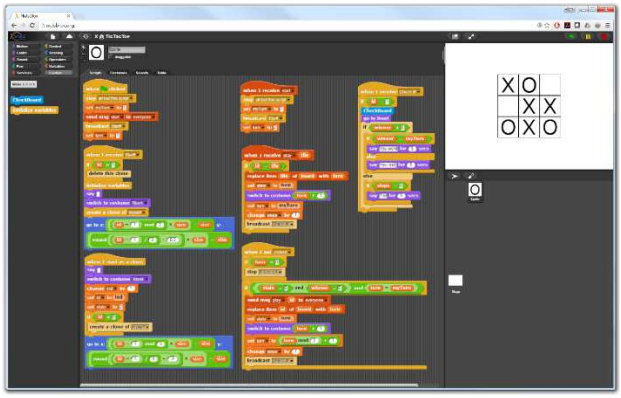

Fig. 6 NetsBlox environment

In an educational environment, OOPP contains three sections (see Fig. 7). Specifically, with the OOPP toolbox (see Fig. 7A) comprising the object-oriented commands, including Classes, Interface, Methods Fields, and Values, students could execute these commands through the drag-and-drop of Blocks in the workspace (Fig. 7B). Besides, the students could also design and generate simple object-oriented code, therefore the system could further create the target code automatically. Fig. 7C is the textbox area of the environment, which is identified in the text code output.

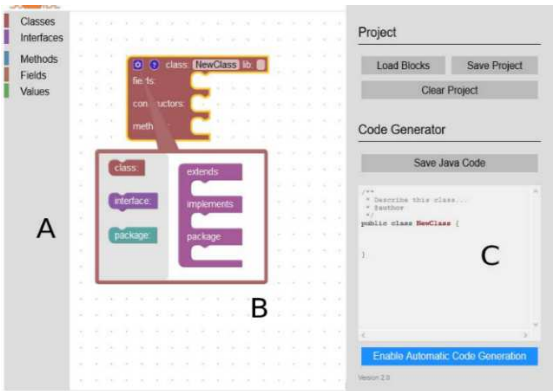

Fig. 7 OOPP environment

The lecturers were interviewed for their opinions about student learning in this study. Besides, their suggestions could be in an interactive programming environment to improve learning programming motivation. Following that, we present an assessment of seven educational learning environments that were criticized based on literature review and issues raised by experts. In this context, relevant literature has been studied and assigned in each of the following environments: the well-known IDE "objects-first" by BlueJ; OOPP hybrid environment; Greenfoot learning and microworld environment; Turtle graphics physically oriented environment; Snap! Game-oriented environment, and the collaboration extension environment of NetsBlox; Alice 3D storytelling and animation environment, and AliCe-ViLlagE virtual pair programming environment.

A. Data collection

The interview was developed to address research questions assessing the issues encountered in previous works and present suggestions to enhance students learning (see Table I). Interviews assist researchers in " getting information about research participants' feelings, thoughts, attitudes, beliefs, attitudes, perceptions, values, personality and behavioral intentions" [53]. The interview consisted of six open-ended questions extracted from previous work to determine the issues faced by the students while learning OOP, including two multiple-choice questions to decide the methods of improving learning programming.

TABLE I
INTERVIEW QUESTIONS

No.	Interview Questions
1	What are the problems faced by the students when learning object-oriented programming?
2	Does the issue of the syllabus place excessive focus on syntax?
3	Is the issue of the syllabus per semester overly broad?
4	Do you agree that several items must be memorized for the reserved word, and students sometimes are faced with difficulty remembering the reserved concepts in the writing program?
5	Do the following issues make object-oriented programming difficult/boring in learning? The complexity of programming concepts The debugging (e.g., find bugs) Insufficient time Absence of interactive media Uncertainty in their solutions Understanding graphics programming (e.g., GUI) Number of students per class/lab
6	Based on your experience, what other issues make object-oriented programming difficult/boring for students?
7	In your opinion, do you agree that the following materials would help in learning object-oriented programming? Interactive visualizations tool Interactive environment Lecture note Programming coursebook
8	In your opinion, do you agree that any of these situations would help in learning object-oriented programming more effectively? In practical Consultation or discussion with lecturers, tutors, seniors, or friends In group exercise sessions While working alone on programming coursework In lectures

B. Data analysis

To address the first research question (RQ1), this study employed the qualitative method via an interview approach to obtain feedback from the interviewees (teachers and lecturers). With an average experience of 10-20 years in learning programming, the interviews were conducted via mobile or face-to-face, recorded, and transcribed under each question.

In addressing the second research question (RQ2) and identifying whether the visual programming environments addressed the issues faced by students in learning OOP, some freely available educational environments were presented and focused on teaching and learning OOP. Particularly, the qualitative criteria of environments were evaluated to define educational suitability as knowledge support for the environments in programming courses.

Before the popularity of the environments of educational programming in the learning and teaching of OOP, the educational systems had existed for a long time. In contrast to today, only a few systems were available in the last century. Accordingly, older systems were straightforward and normally comprised compilers or libraries instead of comprehensive programming environments. However, some changes have occurred in the last decade; educational programming environments are now vital in teaching and learning. Educational programming environments are now more common as they are now considered more acceptable. Similarly, environments have increased further compared to the past. As a result, the design of educational environments has become a topic of great interest.

The turtle graphics environment resembles a turtle; the turtle's pen drives and could be positioned on or off the ground, which leads to the tracking of the turtle's movements. Turtle graphics have been proven highly useful in the teaching of mathematics, particularly the topic of geometry. The basic commands in turtle graphics are easy and simple to learn. Additionally, the actor becomes a virtual turtle in turtle graphics, indicating that it is programmed to relocate and leave "traces" to create various types of drawings. The use of turtle graphics in programming offers fundamental principles of procedural programming, including iteration or recursion [14].

Alice allows students to observe the progress of animated programs immediately. Students could relate the program "piece" to the action seen in animations. In the context of teaching, Alice is an educational interactive programming platform. Alice involves several concepts of programming, including conditional, looping, methods, arrays, parameters, recursion, variables, and basic concepts behind OOP [52]. The environment output is an animated scenario, which is easily understandable among novices. Another extension of Alice is AliCe-ViLlagE, which improves students' confidence in programming by allowing the use of the virtual pair programming approach to solve programming tasks. The result of an empirical investigation [54] proved environmental effectiveness. Despite the advantage of narrative development, the main disadvantages could be seen from the reduced flexibility in the problem-solving domain, absence in a user-friendly interface, and complex environment of the beginner [22].

BlueJ enables the identification of classes and the relationship between them. After the compilation of classes, students could interactively instantiate objects. The main advantage of BlueJ is the apparent separation of the concepts of objects and classes. Although novice students could write

code with BlueJ and draw the hierarchy of the object-oriented solution program, error handling and reporting remain a challenge for the students. Nonetheless, the environment does not offer any visualization representation that assists students' understanding of the concepts of classes and objects. Hence, a teacher still needs to understand the concept of behavior. Besides, the tool does not appear to induce good motivation among learners, thus simply representing the class and object name. The environment does not display visual hints to the condition or state of the object (such as a turtle in Turtle Graphics), which may cause students to become less interested in learning [55].

Greenfoot blends programming in Java with interactive outputs. Given that it is based on BlueJ [44], the structural syntax error could be a challenge to those who are still unfamiliar with the environment and find it challenging to diagnose and fix environment errors. In terms of its motivational linkage, Greenfoot may be more common among students in this field compared to other programming environments presented to date. It could support game creation, as proven by the founders of Greenfoot who have a good understanding of the appeal of games to certain students. However, Greenfoot is more of a "micro-world meta-framework", allowing the generation of diverse micro-worlds, from games to simulations and visualizations, in several areas. One of the advantages of using the environment is the encouragement to increase the difficulty of students' tasks through examples related to gender, cultural background, and age [56]. However, given that Greenfoot does not allow novices to master the concepts of programming languages conveniently, a teacher must fully understand the code and allow the novices to address the challenges [56].

Snap! and NetsBlox. The NetsBlox environment does not allow students to communicate with each other from a distance to discuss problem-solving because it lacks communication features, such as video, voice, or text (appropriate communication). Moreover, given that collaboration features have only been introduced to the tool recently, more studies are required to determine the effectiveness of these features [57].

Multi-Device Grace Co-located collaborative block-based programming, Multi-Device Grace, was constructed to explore Block-programming on all touch-screen devices. The environment allows students to develop and share blocks with various touch devices, mainly tablets. While the environment disables collaboration in real-time, it enables students to operate and distribute a project across devices. Apart from that, the tool is developed only for touch devices [50], and the tool does not present a suggested solution for any error that might occur.

Object-Oriented Puzzle Programming (OOPP) The development of OOPP is based on Google Blockly to support OOP with a strength that represents textual language in Blocks and solves the issues related to writing code using syntax-based languages [45]. Therefore, it may ease the transition to text for some students. Also, in this environment, most Blocks employ standard written sentences while iconic Blocks depend on shapes instead of text to distribute the Blocks' information. However, the drawbacks of OOPP are related to their support of OOP concepts with no basic programming concepts (e.g., conditional, looping), leading to a lack of understanding of the object-oriented concept among students [58]. Another drawback of OOPP is the direct visualization of object state or behavior (e.g., Karel). Thus, the students might not be able to test the code output similarly to BlueJ.

III. Result and Discussion

The results of the research questions are discussed in this section.

A. RQ1: What are the experts' perspectives on the issues (concept difficulty, memorizing, bugs, number of students, and unconfident) leading to the demotivation of OOP learners?

In this question, the interview approach results, and analysis are presented. Interview with experts. The results of the analysis performed with the participants revealed a negative view of the failure/drop rates for programming courses and the impacts of some environments on programming and students' motivation by developing a program linked to their interests (e.g., game and robot). The interviews illustrated the issues and suggestions that should be focused on to avoid the issues encountered by students in the future, with the examples as follows:

1) Interviewee one: "Students are currently having difficulty understanding object-oriented concepts. It is not clear to them; Complexity, remembering the syntax (memorizing the concepts), syntax errors, insufficient time, code quality, and many students per lab are fundamental issues students face while learning programming. Therefore, these difficulties hinder them from enjoying programming and motivate them to develop their programming skills. The essential requirement is to improve their motivation using any technique such as working as groups, robots, games, or challenging them."

2) Interviewee two: "Using the Games/Visualization tools to motivate students in programming a certain period and then move to textual programming is a good idea because students have problems with syntax error and make them more confident. Another problem is the number of students in the laboratory/class, and communication methods with students and between them; therefore, an alternative method is needed."

This section analyses the following information of the interviewees presented to verify the issues:

- All interview feedback participants generally agreed with previous issues (concept difficulty, memorizing, bugs, number of students, and unconfident).
- Some participants suggested another issue, such as communication with/between students.
- All participants agreed that the visualization/interactive environment was the most effective way to improve student learning. It was also decided that the ideal situations to help students learn object-oriented were practice, consultation (discussion with lecturers, tutors, seniors, or friends), and group sessions.

Ultimately, the correlation between previous work and the perspectives from the interview was that all agree with the issues in learning programming, such as memorizing, bugs, complexity or difficulty of concepts, unconfident, and

students' number. Interviewers suggested another issue, which was communication with/between students. In terms of the practical aspects of supporting students, it was recommended that an interactive environment be created to encourage students to practice programming, learn from their peers, and obtain benefit from the teachers' expertise

TABLE II
VISUAL PROGRAMMING ENVIRONMENTS EVALUATION

Environments	**NetsBlox**	**Turtle graphics**	**BlueJ**	**Alice/ AliCe-ViLlagE**	**Greenfoot**	**OOPP**	**Multi-Device Grace**	**Proposed environment**
Approach	Block-based approach		UML-notation approach	Storytelling approach	Game-based approach	Block-based approach	Block-based approach	Block-Based approach
Direct representation	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Syntactical errors	No	Yes	Yes	No	Yes	No	No	No
Text-based representation	No	Yes	Yes	No	Yes	Yes	Yes	Yes
Target group	Primary to university education	Primary and Secondary education	University education	Secondary education (11-15)	University education (14+)	University education	University education	Secondary to university education
Basic programming concepts	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Object-oriented concepts	Yes	Yes	Yes	Objects and method	Yes	Yes	Yes	Yes
Direct error message	No	No	No	No	No	No	No	Yes
Students' community	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Collaborative technique	Yes	Other applications	Other applications	Yes	Other applications	Other applications	Yes	Yes
Virtual pair programming	side-by-side programming	No	No	Yes	No	No	Yes	Yes
Discussion Forums	No	No	No	No	No	No	No	Yes

B. RQ2: Did the seven object-oriented learning programming environments tackle motivation issues?

Several popular environments for learning object-oriented are discussed in this paper in terms of strengths and shortcomings (see Table II), leading to demotivation. Among the analyzed environments, BlueJ and Greenfoot comprised an advantageous feature that allows the inspection and interaction with objects. However, the main drawbacks are the failure to eliminate the likelihood of syntax errors, including the lack of focus on the logic of assigned problems and their solutions. As for novice students, error handling could be challenging and lead to low motivation. Meanwhile, although systems such as turtle graphics offer good object behavior visualization, object interaction is lacking.

Multi-Device Grace, AliCe-ViLlagE, and NetsBlox have recently added collaboration features, such as side-by-side programming, virtual pair programming, or forums to enhance students' confidence and learning of the programming language. Moreover, empirical investigation is required to determine the usefulness and effectiveness of NetsBlox, Multi-Device Grace, and OOPP approaches. The environments offered by Alice, AliCe-ViLlagE, NetsBlox, Multi-Device Grace, and OOPP eliminate the likelihood of syntax errors and encourage students to concentrate on the programming concepts. However, Alice also offers a straightforward and basic representation underpinning OOP concepts, while OOPP eliminates the majority of syntax errors and directly generates a programming language code. Additionally, the OOPP presents an OOP with no basic programming concepts, leading to the students' incapability to understand the correct meaning of object-oriented concepts [58]. Besides, similar to BlueJ, OOPP does not provide direct visualization of object state or behavior (e.g., Turtle). According to Alice and AliCe-ViLlagE, an understandable result was recorded although the problem-solving flexibility was reduced, the design was not a user-friendly and complex environment.

In recent years, different approaches were proposed for dissemination in learning programming. Among many projects aiming to facilitate the introduction of coding, the famous projects code.org, Scratch, the Raspberry Pi platform, and many others could be highlighted [22]. The feature shared by most of these projects was the use of Block Programming to simplify the first approach to programming, reducing and eliminating syntactic difficulties. Meanwhile, block-based programming could reduce a student's cognitive load [30], and encapsulate the code into smaller code chunks to be utilized instead of remembering the code syntax.

In the programming teaching field, the collaborative programming framework was shown to be an efficient learning programming method [59]. Participation in a pair of

microworld projects created a higher sense of accomplishment and confidence in assignments among learners [34], including the discussion on how to solve tasks with others or utilize interactions, assist students in solving complex programming problems and develop programming skills [35]. Therefore, pair programming and discussion forums were used to increase student enjoyment (comfort, insights, interest), motivation, and confidence and improve learning performance [33], [34].

IV. Conclusions

Regarding the general aims of this study, the most common issues faced by students when learning programming was based on previous work and perspectives in interviews, such as memorizing, bugs, the complexity of concepts, unconfident, and the number of students in labs. The necessary skills should be useful in educational environments that support and facilitate object-oriented learning (see Table II). It was found that visualization environments were related to learning and teaching OOP and focused on some issues. However, the coverage of other elements of interest for students to provide the appropriate motivation was lacking. In terms of the previous work supported by the experts' perspective, visualization technique (Block-based approach) and collaborative learning could be important in the execution of learning object-oriented programming to enhance the learning process, which could be challenging, complex and require hard work from the students. Future work could include developing a programming environment that addresses all these issues and measuring the effectiveness, usefulness, and motivation with the usability of learning applications as pre-test and post-test in terms of student learning progress.

Acknowledgment

This research was supported by the Ministry of Higher Education of Malaysia and the GUP-2018-155. The authors appreciate all the parties who have assisted in accomplishing this work.

References

[1] R. Queirós, M. Pinto, and T. Terroso, "Computer Programming Education in Portuguese Universities," *OpenAccess Ser. Informatics*, vol. 81, no. 21, pp. 1–11, 2020, doi: 10.4230/OASIcs.ICPEC.2020.21.

[2] C. S. Cheah, "Factors contributing to the difficulties in teaching and learning of computer programming: A literature review," *Contemp. Educ. Technol.*, vol. 12, no. 2, pp. 1–14, 2020, doi: 10.30935/cedtech/8247.

[3] M. Aissa, M. Al-Kalbani, S. Al-Hatali, and A. BinTouq, "Novice learning programming languages in omani higher education institution (Nizwa University) issues, challenges and solutions," in *Sustainable Development and Social Responsibility—Volume 2*, Springer, 2020, pp. 143–148.

[4] C. Chibaya, "A Metaphor-Based Approach for Introducing Programming Concepts," in *Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019*, 2019, pp. 1–8, doi: 10.1109/IMITEC45504.2019.9015888.

[5] T. W. Koet and A. Abdul Aziz, "Teachers' and Students' Perceptions towards Distance Learning during the Covid-19 Pandemic: A Systematic Review," *Int. J. Acad. Res. Progress. Educ. Dev.*, vol. 10, no. 3, pp. 531–562, 2021, doi: 10.6007/ijarped/v10-i3/11005.

[6] K. V. Zacharis and A. D. Niros, "Computational Thinking," *Commun. ACM*, vol. 49, no. 3, pp. 140–158, 2020, doi: 10.4018/978-1-7998-4576-8.ch006.

[7] R. Hawariyah, B. Z. Halimah, A. Azlina, and M. A. Nazlena, "Student' s Difficulties in Learning Programming," *Adv. J. Tech. Vocat. Educ.*, vol. 2, no. 3, pp. 40–43, 2018.

[8] S. Hodges, S. Sentance, J. Finney, and T. Ball, "Physical Computing: A Key Element of Modern Computer Science Education," *Computer (Long. Beach. Calif).*, vol. 53, no. 4, pp. 20–30, 2020, doi: 10.1109/MC.2019.2935058.

[9] A. V. Robins, "Novice Programmers and Introductory Programming," *Cambridge Handb. Comput. Educ. Res.*, pp. 327–376, 2019, doi: 10.1017/9781108654555.013.

[10] P. O. Jegede, E. A. Olajubu, A. O. Ejidokun, and I. O. Elesemoyo, "Concept-based analysis of java programming errors among low, average and high achieving novice programmers," *J. Inf. Technol. Educ. Innov. Pract.*, vol. 18, no. June, pp. 49–59, 2019, doi: 10.28945/4322.

[11] J. Cuny, "Transforming High School Computing: A Compelling Need, A National Effort." 2012, doi: http://www.worldcat.org/isbn/0071362681.

[12] N. C. C. C. Brown and G. Wilson, "Ten quick tips for teaching programming," *PLoS Comput. Biol.*, vol. 14, no. 4, p. e1006023, 2018, doi: 10.1371/journal.pcbi.1006023.

[13] N. O. Anwar, H. Okumura, T. Widiyaningtyas, and U. Pujianto, "NemU: Design and improvement of visual programming environment as learning support system on basic programming subjects," in *ACM International Conference Proceeding Series*, 2019, vol. Part F1483, pp. 54–61, doi: 10.1145/3323771.3323788.

[14] K. J. Mackin, "Turtle graphics for early Java programming education," *Artif. Life Robot.*, vol. 0, no. 0, p. 0, 2019, doi: 10.1007/s10015-019-00528-y.

[15] N. Fraser, "Google Blockly: A Web-based Visual Programming Editor," *Google*, 2013.

[16] R. Shen, D. Y. Wohn, and M. J. Lee, "Comparison of Learning Programming between Interactive Computer Tutors and Human Teachers," in *CompEd 2019 - Proceedings of the ACM Conference on Global Computing Education*, 2019, pp. 2–8, doi: 10.1145/3300115.3309506.

[17] A. Sartori and C. Schlette, "Visual Programming of a Human-Machine Interface for a Multi-Robot Support System," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2021, pp. 387–392, doi: 10.1109/icps49255.2021.9468200.

[18] H. W. Hnin and K. K. Zaw, "Element Fill-in-Blank Problems in Python Programming Learning Assistant System," in *2020 International Conference on Advanced Information Technologies (ICAIT)*, 2020, pp. 88–93.

[19] A. Adeliyi *et al.*, "Remote Pair Programming," in *SIGCSE 2021 - Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, p. 1289, doi: 10.1145/3408877.3439681.

[20] S. R. Manoharan, T. K. Hua, and F. M. M. Sultan, "A Comparison of Online Learning Challenges Between Young Learners and Adult Learners in ESL Classes During the COVID-19 Pandemic: A Critical Review," *Theory Pract. Lang. Stud.*, vol. 12, no. 1, pp. 28–35, 2022, doi: 10.17507/tpls.1201.04.

[21] M. Mohtar and M. Md Yunus, "A Systematic Review of Online Learning during COVID 19: Students' Motivation, Task Engagement and Acceptance," *Arab World English J.*, no. 2, pp. 202–215, 2022, doi: 10.24093/awej/covid2.13.

[22] S. Xinogalos, M. Satratzemi, and C. Malliarakis, "Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment?," *Educ. Inf. Technol.*, vol. 22, no. 1, pp. 145–176, 2017, doi: 10.1007/s10639-015-9433-1.

[23] M. A. Bakar, M. Mukhtar, and F. Khalid, "The development of a visual output approach for programming via the application of cognitive load theory and constructivism," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 11, pp. 305–312, 2019, doi: 10.14569/IJACSA.2019.0101142.

[24] M. A. Bakar, M. Mukhtar, and F. Khalid, "The effect of turtle graphics approach on students' motivation to learn programming: A case study in a malaysian university," *Int. J. Inf. Educ. Technol.*, vol. 10, no. 4, pp. 290–297, 2020, doi: 10.18178/ijiet.2020.10.4.1378.

[25] J. M. Rodriguez Corral, I. Ruiz-Rube, A. Civit Balcells, J. M. Mota-Macias, A. Morgado-Estevez, and J. M. Dodero, "A study on the suitability of visual languages for non-expert robot programmers," *IEEE Access*, vol. 7, pp. 17535–17550, 2019, doi: 10.1109/ACCESS.2019.2895913.

[26] M. Krafft, G. Fraser, and N. Walkinshaw, "Motivating Adult Learners by Introducing Programming Concepts with Scratch," in *ACM International Conference Proceeding Series*, 2020, pp. 22–26, doi: 10.1145/3396802.3396818.

[27] C. Y. Tsai, "Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy," *Comput. Human Behav.*, vol. 95, pp. 224–232, 2019, doi: 10.1016/j.chb.2018.11.038.

[28] M. Seraj, E. S. Katterfeldt, K. Bub, S. Autexier, and R. Drechsler, "Scratch and google blockly: How girls' programming skills and attitudes are influenced," in *ACM International Conference Proceeding Series*, 2019, p. 23, doi: 10.1145/3364510.3364515.

[29] W. C. Hsu and J. Gainsburg, "Hybrid and Non-Hybrid Block-Based Programming Languages in an Introductory College Computer-Science Course," *J. Educ. Comput. Res.*, vol. 59, no. 5, pp. 817–843, 2021, doi: 10.1177/0735633120985108.

[30] D. Weintrop, "Education block-based programming in computer science education," *Commun. ACM*, vol. 62, no. 8, pp. 22–25, 2019, doi: 10.1145/3341221.

[31] Ł. Wiechetek, "Teaching basic of programming with the elements of Scratch - Evaluation of VBA programming course for logistics students," *Int. J. Innov. Learn.*, vol. 28, no. 2, pp. 159–179, 2020, doi: 10.1504/IJIL.2020.108972.

[32] B. Díaz-Lauzurica and D. Moreno-Salinas, "Computational thinking and robotics: A teaching experience in compulsory secondary education with students with high degree of apathy and demotivation," *Sustain.*, vol. 11, no. 18, p. 5109, 2019, doi: 10.3390/su11185109.

[33] L. K. Lee, T. K. Cheung, L. T. Ho, W. H. Yiu, and N. I. Wu, "Learning computational thinking through gamification and collaborative learning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11546 LNCS, pp. 339–349, doi: 10.1007/978-3-030-21562-0_28.

[34] B. Zhong and T. Li, "Can Pair Learning Improve Students' Troubleshooting Performance in Robotics Education?," *J. Educ. Comput. Res.*, vol. 58, no. 99, pp. 220–248, 2019, doi: 10.1177/0735633119829191.

[35] A. F. Zulfikar *et al.*, "The effectiveness of online learning with facilitation method," *Procedia Comput. Sci.*, vol. 161, pp. 32–40, 2019, doi: 10.1016/j.procs.2019.11.096.

[36] M. Satratzemi, D. Tsompanoudi, S. Xinogalos, and L. Karamitopoulos, "Examining the compatibility of students in distributed pair programming," in *Proceedings of the European Conference on e-Learning, ECEL*, 2019, vol. 2019-Novem, pp. 510–518, doi: 10.34190/EEL.19.023.

[37] J. Hughes, A. Walshe, B. Law, and B. Murphy, "Remote pair programming," in *CSEDU 2020 - Proceedings of the 12th International Conference on Computer Supported Education*, 2020, vol. 2, no. Csedu, pp. 476–483, doi: 10.5220/0009582904760483.

[38] Y.-C. Hsu, Y.-H. Ching, J. Callahan, and D. Bullock, "Enhancing STEM Majors' College Trigonometry Learning through Collaborative Mobile Apps Coding," *TechTrends*, vol. 65, no. 1, pp. 26–37, 2021.

[39] D. Tsompanoudi, M. Satratzemi, S. Xinogalos, and L. Karamitopoulos, "An Empirical Study on Factors related to Distributed Pair Programming," *Int. J. Eng. Pedagog.*, vol. 9, no. 2, pp. 65–81, 2019, doi: 10.3991/ijep.v9i2.9947.

[40] C. M. C. De Oliveira, E. D. Canedo, H. Faria, L. H. V. Amaral, and R. Bonifacio, "Improving Student's Learning and Cooperation Skills Using Coding Dojos (In the Wild!)," in *Proceedings - Frontiers in Education Conference, FIE*, 2019, vol. 2018-Octob, pp. 1–8, doi: 10.1109/FIE.2018.8659056.

[41] J. Cheng and J. Lei, "A description of students' commenting behaviours in an online blogging activity," *E-Learning Digit. Media*, vol. 18, no. 2, pp. 209–225, 2021, doi: 10.1177/2042753020954971.

[42] B. Laurel, "Computers as Theater-Laurel, B." Visible Language RISD-Graphic Design Dept 2 College ST, Providence, RI 02903, 1993.

[43] C. Jost and B. Le Pévédic, *Designing Evaluations: Researchers' Insights Interview of Five Experts*. Springer International Publishing, 2020.

[44] M. Kölling, "Blue, bluej, greenfoot: Designing educational programming environments," in *Innovative Methods, User-Friendly Tools, Coding, and Design Approaches in People-Oriented Programming*, IGI Global, 2018, pp. 42–87.

[45] A. Ferrari, G. Lombardo, M. Mordonini, A. Poggi, and M. Tomaiuolo, "OOPP: Tame the Design of Simple Object-Oriented Applications with Graphical Blocks," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 2018, vol. 233, pp. 279–288, doi: 10.1007/978-3-319-76111-4_28.

[46] D. Garcia and M. Ball, "Snap! 6, Introducing Hyperblocks!," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, p. 1379.

[47] G. Stein and A. Lédeczi, "Enabling Collaborative Distance Robotics Education for Novice Programmers," in *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2021, pp. 1–5.

[48] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts," *J. Comput. Sci. Coll.*, vol. 15, no. 5, pp. 107–116, 2000.

[49] A. Al-Jarrah and E. Pontelli, "AliCe-ViLlagE Alice as a Collaborative Virtual Learning Environment," in *Proceedings - Frontiers in Education Conference, FIE*, 2015, vol. 2015-Febru, no. February, pp. 1-9. doi:10.1109/FIE.2014.7044089, doi: 10.1109/FIE.2014.7044089.

[50] B. Selwyn-Smith, C. Anslow, M. Homer, and J. R. Wallace, "Co-located Collaborative Block-Based Programming," in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2019, pp. 107–116.

[51] B. T. Fasy, S. A. Hancock, B. Z. Komlos, B. Kristiansen, S. Micka, and A. S. Theobold, "Bring the page to life: Engaging rural students in computer science using alice," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 110–116.

[52] C. Kelleher and R. Pausch, "Using storytelling to motivate programming," *Commun. ACM*, vol. 50, no. 7, pp. 58–64, 2007, doi: 10.1145/1272516.1272540.

[53] B. Johnson and C. Larry, *Quantitative, Qualitative, and Mixed Approaches*. SAGE Publications, Incorporated, 2003.

[54] A. Al-Jarrah and E. Pontelli, "The collaborative virtual affinity group model: principles, design, implementation, and evaluation," *Int. J. Comput. Appl.*, vol. 42, no. 5, pp. 485–513, 2020.

[55] M. Kölling, "Lessons from the Design of Three Educational Programming Environments," *Int. J. People-Oriented Program.*, vol. 4, no. 1, pp. 5–32, 2015, doi: 10.4018/IJPOP.2015010102.

[56] S. J. Cox and S. J. Johnston, *Raspberry Pi Technology*. MDPI, 2018.

[57] N. Lytle, A. Milliken, V. Catete, and T. Barnes, "Investigating different assignment designs to promote collaboration in block-based environments," in *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 2020, pp. 832–838, doi: 10.1145/3328778.3366943.

[58] J. Lewis, "Myths about object-orientation and its pedagogy," in *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, 2000, vol. 32, no. 1, pp. 245–249, doi: 10.1145/331795.331863.

[59] A. L. Asnawi *et al.*, "The needs of collaborative tool for practicing pair programming in educational setting," *Int. J. Interact. Mob. Technol.*, vol. 13, no. 7, pp. 17–30, 2019, doi: 10.3991/ijim.v13i07.10722.