IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 June 2024, accepted 8 August 2024, date of publication 15 August 2024, date of current version 4 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3443879

RESEARCH ARTICLE

A Choice-Based Programming Learning Method to Develop Problem-Solving Skills

HIROKI OKA[1], AYUMI OHNISHI[1], TAKESHI NISHIDA[2], TSUTOMU TERADA[1], (Member, IEEE), AND MASAHIKO TSUKAMOTO[1], (Member, IEEE)

[1]Graduate School of Engineering, Kobe University, Kobe, Hyogo 657-8501, Japan
[2]Graduate School of Intercultural Studies, Kobe University, Kobe, Hyogo 657-0011, Japan

Corresponding author: Tsutomu Terada (tsutomu@eedept.kobe-u.ac.jp)

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Ethical Review Committee for Research Directly Involving Human Subjects of Graduate School of Engineering, Kobe University under Permission Nos. 05-18 and 05-29.

ABSTRACT Programming is becoming a new literacy, and programming education is being implemented in various settings. One of the issues faced by novice programmers is a lack of problem-solving skills. They have difficulty planning abstract solutions and expressing them in a program. This study proposes a programming learning system that supports novice programmers' thinking processes when solving programming tasks. The system presents candidates for solutions in natural language and allows the user to solve the problem by choosing from a limited number of options. The method allows users to repeat quick trial-and-error solutions without being interrupted by coding-induced problems. We implemented the proposed system as a web application for use on a smartphone and applied it in a university's introductory programming class. After applying the proposed system to 37 students for four weeks, we found a correlation between the number of times a user selected an option using the proposed system and the class task score. We confirmed the possibility that using the proposed system can improve problem-solving skills for programming tasks.

INDEX TERMS Programming, programming education, end-user programming, problem-solving, exploratory programming.

I. INTRODUCTION

Programming is becoming a new form of literacy worldwide, and computers are widely integrated into daily life. Programming education is now offered in elementary schools as part of the development of computational thinking [1] and STEM education [2], and services to support programming learning are growing [3], [4], [5]. In addition, End-User Development (EUD) [6], [7] environments are proliferating, allowing non-computer experts to enjoy the benefits of programming. Various types of programming support are available using large language models (LLM) such as ChatGPT [8], [9]. Learning programming is becoming more critical, and the fields in which programming can be utilized are expanding.

On the other hand, novice programmers face many barriers to learning programming. Many studies have been conducted on the difficulties faced by novice programmers when learning programming. In most previous studies, understanding the abstract concepts and syntax necessary for programming, designing programs to solve problems, and analyzing problems are often cited as difficulties for them [10], [11], [12], [13], [14], [15]. Our study focuses on the problem-solving skills of novice programmers in programming. Problem-solving skills are crucial in learning programming [16], [17]. The problem-solving process in programming generally involves four steps: understanding the problem, planning a solution, implementing the solution, and testing and debugging the solution [18]. To improve problem-solving skills in programming, it is necessary to repeat the four steps through repeated trial and error. However, novice programmers are hindered by a lack of

The associate editor coordinating the review of this manuscript and approving it for publication was Antonio Piccinno.

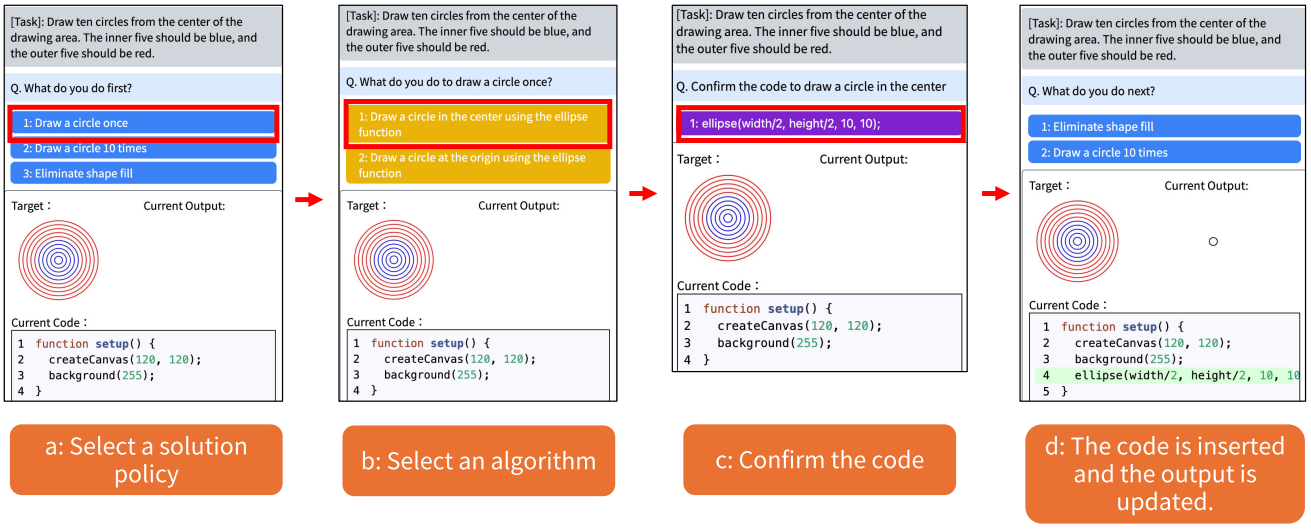

FIGURE 1. Example of system usage in solving a programming task.

programming understanding, and troubles caused by the code, such as implementation and syntax errors, make the trial-and-error process time-consuming.

This study proposes a programming learning method that supports novice programmers' thinking processes when solving programming tasks. This method presents candidate solutions for programming tasks in natural language rather than in a program. It improves problem-solving skills in programming by allowing novice programmers to rapidly trial and error various candidates of solution alternatives for programming tasks. We implemented the proposed method as a web application that can be used from a smartphone.

In addition, the system separates how to plan a solution and code a solution and the user can focus on planning a solution in learning programming. The system automatically suggests the corresponding program code when the user selects the solution candidate for the given programming task.

The target environment is a programming learning environment for beginners using visual output from program execution, such as Processing [19] or p5.js [20]. The system supports developing programming learners' problem-solving skills while it does not support developing programming skills itself. Therefore, the system is intended to be used in parallel with classes on programming syntax and self-study.

Figure 1 shows an example of the usage of the proposed system in solving the programming task. In this figure, the task is to write code that draws ten circles, painting the outside in red and the inside in blue. The system displays the options for the solution policy of the task in natural language, and the user selects the first option (a). Next, the user selects the option of an algorithm that can implement the chosen option before (b). The system displays a code for the chosen option (c). When the user confirms the code, the system will update the current code and output (d). Repeating these steps allows the user to concentrate on planning a solution.

In this paper, we implemented the proposed method as a system and conducted an evaluation experiment in a university introductory programming course. Contributions of this paper can be summarized as follows:

- We propose a new programming learning method based on selecting options for solutions in natural language to improve the problem-solving skills of novices. The proposed method presents candidates for solutions in natural language and allows users to repeat quick trial-and-error solutions without being interrupted by coding-induced problems.
- The proposed method is implemented as a user-friendly web application, ensuring easy access for users, even from their smartphones.
- We applied the system in an actual programming class. As a result, the proposed system can potentially increase novices' lines of code and improve their programming task scores.

II. RELATED WORK

In this section, we describe the prevalence of programming environments for end-users and the difficulties that novice programmers face in learning programming. We present previous research on programming learning approaches for novices and describe exploratory programming, an effective method for them to learn.

A. PROGRAMMING ENVIRONMENT FOR NON-EXPERTS

Computers have become ubiquitous in our daily lives, and more and more tools are available for people to utilize programs even if they are not programming experts. There are various approaches to end-user programming and EUD [6], including visual programming and example programming. Scratch [21], LOGO [22], and Alice [23] are examples of visual programming languages that are widely used in

programming education. Exemplary programming [24] is a programming approach that aims to program by illustrating data movement to the computer.

EUD environments that incorporate live programming have been proposed. This approach removes the barriers between editing, executing, and debugging the code and provides continuous feedback. Lau et al. proposed Tweakit, a tool that allows data analysts to easily preview the behavior of Python code live in Excel to support data analysis using their code [7]. Kato and Goto proposed an approach that offers two types of interfaces within a single Integrated Development Environment (IDE) based on the user's programming level [25]. They offer a live programming interface for skilled programmers to directly edit the code and a live tuning interface for end-users to manipulate code parameters using a GUI, such as sliders.

In addition to these EUD approaches, some studies have explored new programming approaches. Krings et al. analyzed software development scenes depicted in science fiction films and discovered concepts for EUD, including Programming by Physical Interaction, Programming by Natural Language, and Emotional Programming [26]. Kato and Shimakage emphasized the importance of collaboration among individuals with varying programming skills as computers become more ubiquitous [27]. They also stressed the importance of developing interfaces for diverse users and understanding the socio-technical aspects of domain-specific applications.

Therefore, numerous programming environments and approaches for end-users continue to emerge and expand. However, even end-users must work with abstract concepts on the computer and understand programming to use programs in their fields. It is difficult for novices to tackle open-ended problems in the program out of the box.

B. DIFFICULTIES IN LEARNING PROGRAMMING

The difficulty of learning programming has been widely investigated. Derus and Ali conducted a questionnaire on the difficulty of learning programming among students taking an introductory programming course [10]. The results showed that the most difficult learning content was abstract concepts such as multidimensional arrays, iterations, and functions, understanding the structure of programs, and designing programs to solve problems. Lahtinen et al. also surveyed over 500 students and teachers on the difficulties of learning programming. They found that the most difficult concepts were abstract concepts such as recursion, pointers and references, and abstract data types, and the most difficult tasks were ''understanding how to design a program to solve a problem,'' ''breaking a function into steps,'' and ''finding bugs in the program'' [11]. Ismail et al. interviewed five computer science teachers to identify the leading causes of problems students face in learning programming [12]. The study found that the four main causes of these problems are: (1) lack of problem analysis skills, (2) ineffective problem-solving representations, (3) ineffective teaching methods for problem-solving and programming, and (4) insufficient ability to understand and master the syntax and structure of programs.

Lister categorized the stages of programming learning into four categories based on Piaget's stages of cognitive development. According to Lister, in the later stages, students can fully comprehend the structure and design of programs and use abstraction [14]. They also suggest that learners should be allowed to learn new concepts gradually. According to Bain and Barnes barriers to learning programming are related to problem-solving strategies and emotional issues stemming from past educational experiences [13]. Ko et al. identified six categories of learning barriers for end-user development (EUD) related to the programming environment and libraries: design, selection, coordination, use, understanding, and information [15].

Various barriers to learning programming have been identified, including a lack of understanding of grammar and concepts, problem-solving skills, and the cognitive load of learning these skills simultaneously. In this study, we focus on improving the problem-solving skills of novice programmers.

C. PROGRAMMING LEARNING APPROACH FOR NOVICES

As described in the previous section, learning programming can be challenging. In response, various programming learning approaches have been proposed so far.

Brusilovsky et al. classified programming learning approaches for novice programmers into three types: (1) The incremental approach, (2) The mini-language approach, and (3) The sub-language approach [28]. The incremental approach is a step-by-step learning method in which novices learn the basics of programming at the beginning and are gradually taught new concepts. DrScheme [29] is an example. The mini-language approach is a learning method through a simple programming language designed for educational purposes. Examples of such languages include Scratch [21], LOGO [22], and Alice [23]. The sub-language approach is a learning method using a comprehensible subset of a programming language. This includes Processing [19], which is easy to handle visual output, and MiniJava [30], a simple subset of Java.

Other systems have been proposed that incorporate new programming learning approaches. Hermans focused on teaching programming language syntax and proposed a ''gradual language'' format with step-by-step language specification changes. This approach was implemented in the programming language Hedy [31]. Dietz et al. proposed StoryCoder, an audio-guided smartphone application that teaches computer concepts through storytelling to teach computational thinking to children [32]. Shi et al. proposed Pyrus, a programming game designed to enhance the problem-solving skills of novice programmers working in pairs [18].

While several learning programming environments have been suggested, only a few support programming problem-solving [33].

D. EXPLORATORY PROGRAMMING

Exploratory programming is a programming technique in which developers derive requirements by implementing, testing, and debugging programs that satisfy basic requirements when the specifications are unclear [34]. This trial-and-error process of writing test programs is also essential in learning and prototyping.

Kery and Myers organized the history of exploratory programming based on previous research and identified its properties [35]. According to the authors, exploration is essential when learning to perform a task that has never been done before, working on a creative task, or working on a difficult task where the means to achieve the goal is unknown without experimentation. They argue that exploratory programming is a practice that supports creative and open-ended programming tasks. Angert et al. proposed Spellburst, an interface for creative coding using exploratory programming [36]. Spellburst is a node-based visual programming environment with prompts that allow text input to add output from a generative AI. It supports artists' exploratory and creative coding.

Based on the elements of exploratory programming, our study proposes a method for solving programming problems that facilitate trial-and-error for novice programmers.

III. PROPOSED SYSTEM

This section describes a programming learning support system that presents candidates for solutions in natural language and allows the user to solve the problem by choosing from a limited number of options.

A. TARGET

First, we describe the target of the proposed system. The proposed system helps novice learners learn how to plan a solution in programming. The proposed system focuses novices on planning problem-solving solutions by eliminating the coding process. Therefore, the system is assumed to be used parallel with learning syntax and meaning in classes or self-study.

The target users are novice programmers, and the system targets programming learning using visual feedback of program execution, a typical programming environment for novice programmers. This study focuses on learning with p5.js. p5.js is designed to write the code with visual output in a relatively straightforward manner with minimum. For the program that draws the circle in Figure 1, the p5.js program can be represented as the following program 1.

We implemented the proposed system as a web application so users can easily use it from their smartphones. Although this system can be used in the classroom, in this paper, we apply it not for classroom use but for self-study outside of class.

program 1: Draw circles

```
function setup() {
  createCanvas(120, 120);
  background(255);
  noFill();
  for(let i = 0; i < 10; i++){
    let d = (i + 1) * 10;
    if (i < 5) {
      stroke(0, 0, 255);
    } else {
      stroke(255, 0, 0);
    }
    ellipse(width/2, height/2, d, d);
  }
}
```

B. DESIGN POLICY

To assist novice programmers in planning solutions to their problems, we implemented the proposed system based on the following design policies.

1) LET THE USER THINK OF A SOLUTION BY SELECTING FROM A LIMITED NUMBER OF OPTIONS

For novice programmers with little experience in problem-solving programming, programming is so open-ended that novice programmers often need guidance on where to start thinking. Therefore, the proposed system supports the planning of problem-solving tasks by allowing novice programmers to explore limited options.

2) LET THE USER THINK OF A SOLUTION IN A NATURAL LANGUAGE

It is difficult for novice programmers with a limited programming understanding of program syntax and semantics to solve a problem from scratch. The proposed system lets novice programmers think about how to solve the problem by selecting options in natural language. In addition, the system allows the user to focus on solution planning by automatically suggesting the appropriate code when planning a solution through natural language options.

3) ENCOURAGE THE USER TO REFLECT ON THE PLANNED SOLUTION

The thought process for solving a coding problem and the flow of the resulting code does not necessarily match. Therefore, it is difficult for novice programmers to reflect on their thought process used to solve the problem. The proposed system implements a tab that displays the solution options selected by the user in the past to encourage reflection on the solution.

4) PROVIDE SCAFFOLDING FOR CODING

The purpose of the proposed system is to help novice programmers plan solutions. After learning how to plan a solution, users are expected to learn how to code it using a programming language.. We designed a step in which the user selects a solution and confirms how the program will be

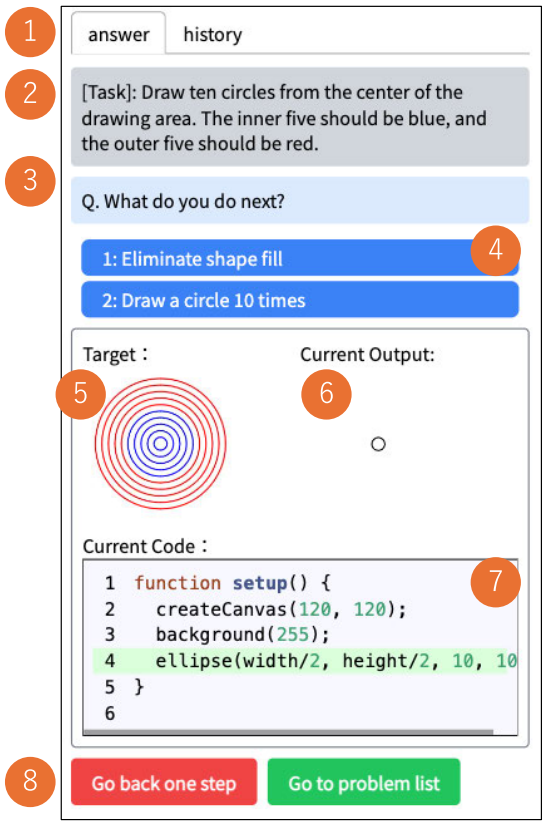

FIGURE 2. User interface.

implemented to provide a scaffold for coding without using the proposed system.

C. USAGE

Figure 2 shows the user interface of the proposed system. When using the system to solve a problem, the user first reads the problem statement (2). The user selects the option for a solution (4) from the displayed options according to the messages (3). As the user selects, the Target (5), Current Code (7), and Current Output (6) on the interface changes, and the user checks them as they solve the problem. If the user feels a selection is incorrect, they can return by clicking the button at the bottom of the screen (8).

The system encourages step-by-step solution planning by having the user solve the problem by repeating the three steps shown in Figure 3.

In the first step, "Policy Selection" (Figure 3(a)), the user selects an abstract policy for what to do first to solve the problem. For example, "draw a circle" or "draw multiple lines using repetition." The options are shown in blue.

In the second step, "Algorithm Selection" (Figure 3(b)), the user selects a procedure or an algorithm to program an abstract policy for solving the problem. For example, "Draw a circle in the center of the canvas using the circle drawing function" or "Draw nine lines using the loop variable." The options are shown in yellow.

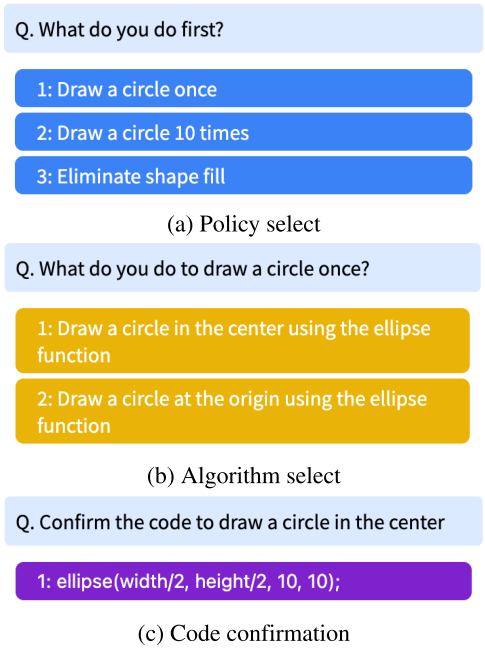

FIGURE 3. Problem-solving steps in the proposed system.

In the third step, "Code Confirmation" (Figure 3(c)), the code corresponding to the choice made in the algorithm selection is suggested and confirmed by the user. This step displays only one code and does not have multiple options. When the user selects a code, it is inserted and highlighted in the system's text editor. This step allows the user to see what the code will look like when the planned solution is actually coded. The code are displayed in purple.

Policy and algorithm selection support planning a solution by showing a limited number of options for rapid trial and error. Code confirmation scaffolds the actual coding.

There are two tabs (1) at the top of the screen, "Answer" and "History." The user proceeds to solve the tasks in the Answer tab and selects the History tab to reflect their strategy. The History tab displays a list of previous choices made by the user, as shown in Figure 4. The choices for each step are displayed hierarchically with color and indentation to encourage the user to look back at the solution. The exact history is displayed in the Answer tab when a problem is solved.

D. IMPLEMENTATION

The proposed system was implemented as a web application using Next.js [37]. The options for each task on the system are expressed in JSON format, which the first author manually prepared.

IV. LABORATORY EXPERIMENT

Before introducing the proposed system to an educational setting, we conducted an in-laboratory experiment to investigate its effects on users' problem-solving skills and usability.

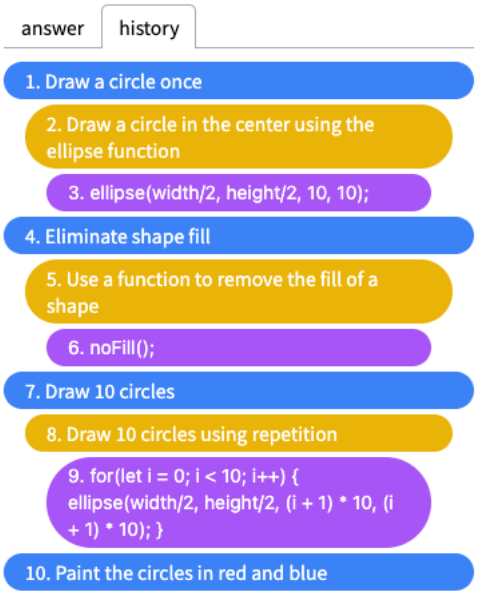

FIGURE 4. History tab.

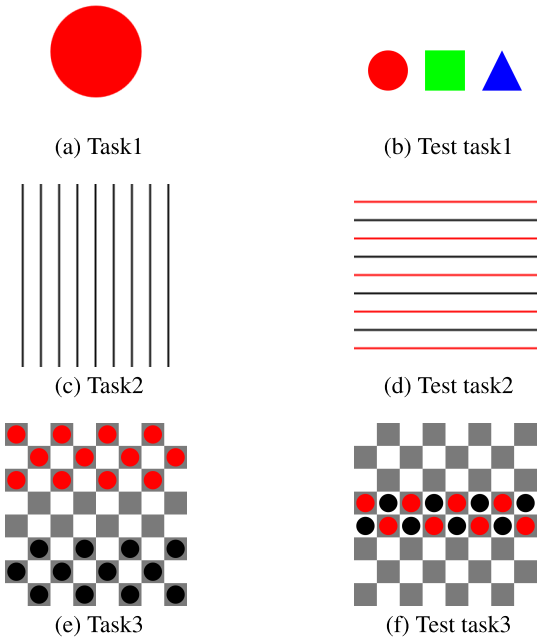

FIGURE 5. Images used in the task and test.

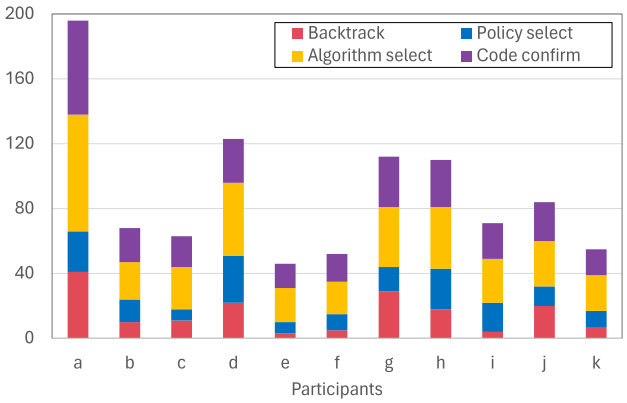

FIGURE 6. Distribution of choices in laboratory experiments.

The experiments described in this section were conducted with the approval of the Ethical Review Committee for Research Directly Involving Human Subjects of the Graduate School of Engineering, Kobe University (approval numbers 05-18).

A. EXPERIMENT DETAILS

The experiment aims to evaluate whether the proposed system can assist novice programmers in planning solutions. At first, participants read texts from an introductory programming course to understand the basics of solving programming tasks in the experiment. After solving the three programming tasks on the proposed system, the participants are tested on their ability to describe the solution procedures in natural language for another three programming tasks.

The participants were 11 high school and college students in their teens and twenties (one female, ten male). Before the experiment, we conducted a questionnaire about the participants' programming experience. Two of the participants answered "never," five answered "I have studied programming on my own or in classes," three answered "I can implement programs using basic concepts such as variables and iteration," and one answered, "I have developed software before." Because the participants were Japanese speakers, all text in the proposed system was displayed in Japanese. The Human-Computer Interaction researchers and the course teacher confirmed the questionnaire items.

The test consists of planning the program that produces a given image. Because the system does not support coding, we do not ask participants to code their solutions. Instead, participants were asked to write their solutions in bullet points. Figure 5 shows the images of the task to be solved by the proposed system and the images to be produced in the test.

We recorded the use of the system to investigate whether the participants engaged in trial-and-error and reflected on their choices. We also evaluated whether participants' problem-solving planning skills had improved based on the test results. After the test, we surveyed the participants about their impressions of the system and its usability.

B. EXPERIMENT RESULT

1) USAGE LOG

We describe the results of analyzing the usage logs. The distribution of selections is shown in Figure 6. Across all participants, the "Go back one step" button was used an average of 15.5 times, indicating that participants selected multiple options through trial and error. Participants reflected their choices on the History tab an average of 3.72 times. Two participants never used the History tab.

- Change the paint color to red.
- Use the function to draw a circle along with coordinates to draw it on the left side.
- Change the paint color to green.
- Use the function to draw a square along with coordinates to draw it in the center.
- Change the paint color to blue.
- Use the function to draw a triangle along with coordinates to draw it on the right side.

FIGURE 7. Solution example for test task 1.

- Repeat 9 times using a for loop (define a variable i, and change the drawing position y of the line with each iteration).
- Conditional branching with if:
  - If the remainder of i divided by 2 is 0: draw a red line (change the paint color to red and draw the line).
  - If the remainder of i divided by 2 is 1: draw a black line (change the paint color to black and draw the line).

FIGURE 8. Solution example for test task 2.

2) TEST RESULT

Examples of participants' solutions for the test are shown in Figures 7, 8, and 9. The test results and the average number of lines per test task are shown in Table 1. The first author interpreted the meaning of the answers and scored them. The scoring criteria were whether the algorithm could be written to output the correct image and whether the correct result could be obtained if the answers were programmed.

Table 1 shows that many participants could describe their solutions step by step. Regardless of their programming experience, some used variables, if and for statements, and indentation to express the block structure.

Most of the incorrect answers were not due to wrong solution policies but to errors in programmatic aspects such as the number of iterations and loop conditions. These results suggest that combining the proposed system with syntax learning support, such as lecture learning, may improve problem-solving skills.

3) QUESTIONNAIRE RESULT

Table 2 shows the questionnaire results about the prototype of the proposed system. From the results of Q1, the proposed system may improve the user's skill to formulate a solution to a programming problem. In the free description, there were some positive comments: "I felt that thinking about what to do next in natural language is useful regardless of the level of one's programming skills in terms of making a solution policy."

- Create a loop using a for statement (set i=0, and as long as i is less than 8, increase i by 1).
- Inside the created for statement, create another for statement (set j=0, and as long as j is less than 8, increase j by 1).
- When the remainder of i+j divided by 2 is 1, specify the color of the shape (gray) and create a square (with x-axis as i and y-axis as j), and remove the outline of the square.
- When the remainder of i+j divided by 2 is 1 and j is greater than 2 but less than 5, specify the color of the shape (red), create a circle (with x-axis as i and y-axis as j), and remove the outline of the shape.
- When the remainder of i+j divided by 2 is 0 and j is greater than 2 but less than 5, specify the color of the shape (black), create a circle (with x-axis as I and y-axis as j), and remove the outline of the shape.

FIGURE 9. Solution example for test task 3.

TABLE 1. Test Result.

	Number of correct answers	Number of lines
Test task1	7	7.73
Test task2	8	4.25
Test task3	3	6.88

On the other hand, a participant with programming experience commented, "I have done this level of problem many times in C, so I did not feel that my skills improved much." The proposed system may not be effective for users who have experience with other programming languages and can already plan solutions.

The results of Q2 - Q4 indicate that the proposed system supports step-by-step solution planning and may allow users to perform trial-and-error more efficiently rather than directly solving programming tasks.

The results of Q5 and Q6 indicate that the proposed system is easy to use. However, some participants pointed out that it is difficult to use. Specifically, one participant commented on the user interface: "At first, I had trouble understanding where the questions were, so it would be easier if the tabs for tasks and answers were reversed." The usability needs to be improved by adding information to be displayed in tabs or changing the default tabs.

Based on the results of TQ1 and TQ2, which are questionnaires on test tasks, the proposed system may enable users to formulate a step-by-step solution strategy when solving program tasks.

The following remarkable comments were also obtained:

- "I thought it was good to see the process of building up the target object gradually. Beginners would appreciate the ability to select the code from options while checking it in text."
- "I felt that I was able to apply the knowledge about the steps of program structure learned through the web app."
- "Before starting the program, I felt that the stage of organizing one's thoughts was clearly recognized."

TABLE 2. Questionnaire Result (1: totally disagree - 5: strongly agree).

Code	Question	AVE	SD
Q1	Do you feel that your ability to develop a strategy for solving problems has improved?	3.64	0.92
Q2	Were you able to think about solutions step by step?	4.45	0.69
Q3	Were you able to try out different options easily?	4.45	0.82
Q4	Was it easier to do the tasks in class?	4.64	0.50
Q5	Were the number and granularity of options in the application appropriate?	3.82	1.33
Q6	Was the application easy to use?	3.18	0.87
TQ1	Were you able to think step by step when solving the test task?	4.09	0.54
TQ2	Did you feel the influence of the proposed system while solving the test task?	4.91	0.30

- "I would like to use it if it existed (as a service). It was an app that made me understand the importance of verbalizing each step towards completion, something we tend to do unconsciously in our usual practice."

Therefore, the proposed system can promote the design and division of solutions to problems before they are tackled.

Based on these results, the proposed system has the potential to support the planning of solutions to programming problems and was highly rated for its usability, suggesting that it can be used in an educational setting.

V. EMPIRICAL EXPERIMENT

We operated the proposed system in an introductory programming course at a university to evaluate whether it could improve the problem-solving skills of beginning students. The experiments described in this section were conducted with the approval of the Ethical Review Committee for Research Directly Involving Human Subjects of the Graduate School of Engineering, Kobe University (approval numbers 05-29).

A. EXPERIMENT DETAILS

We applied the proposed system to an introductory programming course, "Basic Programming Exercise 1," at Kobe University in 2023 and evaluated whether it improved students' problem-solving skills. The course is offered in the third quarter of the Faculty of International Human Sciences of Kobe University. According to the syllabus, the course aims to provide students with an understanding of basic programming concepts, the ability to create basic programs independently, and explain the programs they create clearly. The programming languages and libraries covered in the lectures are JavaScript and p5.js. Students learn basic programming concepts by creating programs with visual output.

The lectures were held eight times in a face-to-face format. After the initial guidance, students work on two to four programming tasks that test the basic programming concepts in the 2nd to 5th lessons. In the sixth to eighth lessons, the students work on a free production task using the knowledge they have learned in class.

The number of students attending the lecture was 41. In the first lecture, we explained to the students how to use the proposed system and that using it would not affect their grades. The lectures were conducted in a flipped classroom style. Students are expected to prepare for the lecture materials before each lecture and work in groups of three to cooperate on tasks during the lecture. The teacher and teaching assistants will grade each task on a three-point scale. The tasks for each lecture are as follows:

- **Lecture 2: Drawing shapes with calculations**
  1) Draw ten circles from the center of the drawing area. The inner five should be blue, and the outer five should be red.
  2) Draw a chessboard.
  3) Draw a dartboard.
  4) Draw a national flag (which contains a repetition structure) of your choice.
- **Lecture 3: Animation and interaction**
  1) Create an animation where the size of a circle changes like a beating heart. Additionally, make the heart beat faster when a keyboard key is pressed and slower when it is released.
  2) Create a program with interaction like a 2D game. Draw a circle that can move left and right on the ground using the left and right arrow keys. Also, make the movement faster while holding down the left or right arrow key and another choice key.
- **Lecture 4: Handling multiple values together**
  1) Calculate an array's average, minimum, and maximum values.
  2) Draw the values calculated in 3-1 as a bar graph.
  3) Create an animation in which balls of random size appear at the spot clicked with the mouse.
- **Lecture 5: Refactoring long programs with functions**
  1) Use a function to draw the flag of the EU.
  2) Create a function that returns how many days are in a given year.

We provided the proposed system as one of the preparation materials to be used before class. After reading the class text, students were instructed to solve a problem for each class using the system. We prepared six programming tasks in the proposed system. One task is to check the proposed system's operation and create a program to draw the Japanese flag. The other five questions were the same as the class tasks 2-1, 2-2, 3-1, 4-1, and 5-1. As in the laboratory experiment, all text in

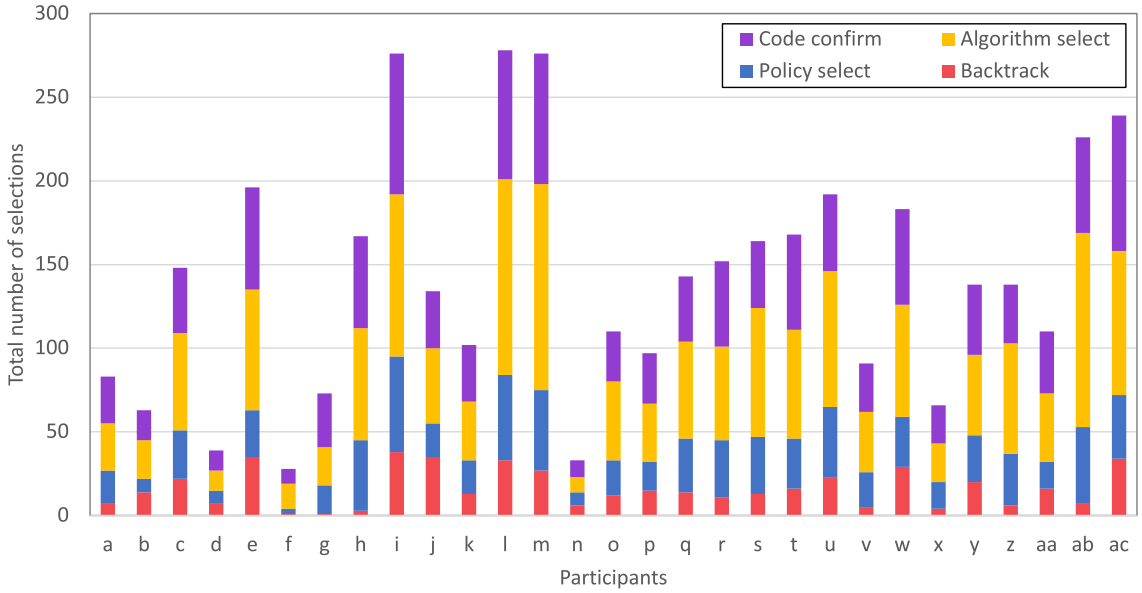

FIGURE 10. Distribution of choices in empirical experiments.

TABLE 3. Number of times each feature was used.

	AVE	SD
Choices	125.7	64.1
Backtrack	16.1	11.4
History Tab	1.90	2.81
Completed tasks	4.52	1.62

TABLE 4. Results of class task efforts.

	AVE	SD
Total score	26.6	6.72
Number of submissions	9.03	2.04

the proposed system was also displayed in Japanese in this experiment.

We logged how the participants used the proposed system. We also collected the programs of the class tasks and analyzed what kind of programs the participants wrote. In the final lecture, we conducted a questionnaire to the participants about their use of the system. The Human-Computer Interaction researchers and the course teacher confirmed the questionnaire items. Responses to the questionnaire were optional.

VI. EMPIRICAL EXPERIMENT RESULT

A. USAGE LOG

First, we describe the participants' usage logs of the proposed system. Students who did not enter their names when using the system were excluded from the analysis. We analyzed the data of 29 participants (18 females and 11 males).

The usage log of the proposed system is shown in Table 3. The number of tasks completed in Table 3 indicates that many participants worked on more than half of the tasks and tried several options by backtracking. Figure 10 shows the number of choices made by each participant and their distribution. Figure 10 indicates that the number of choices and their breakdown varied from participant to participant.

Figure 11 shows how one participant explored the options for solutions in Task 2-1. In Figure 11, each node represents a choice, and the black edges represent nodes that can be transitioned by the choice. The color of a node indicates its state, such as Policy or Algorithm. The red arrows indicate the user's path to reach the correct answer. In Figure 11, the Code confirm step is omitted because it is paired with the Algorithm select step. This is an example of a use case in which the participant explores the options in trial and error.

We have analyzed the relationship between the use of the proposed system and the class tasks. Table 4 shows the participants' class task efforts. Many participants worked on more than half of the class tasks. There was no correlation between the number of backtracks and the class task score, and we could not confirm a trend that the more backtracks participants made, the more trial-and-error they made, and the higher their class task score.

We calculated Pearson's product rate correlation coefficient between the total number of choices and the class task score in the proposed system and found a weak correlation ($r = 0.313$, $p = 0.098$). We grouped participants according to the number of choices they made using the proposed system and compared the class task scores of the groups' top 25% (7 participants) and bottom 25% (participants) regarding the number of choices made using one-factor within-participant analysis of variance. The results are shown in Figure 12, and there was a significant trend in the class task scores between the groups ($F_{(1,12)} = 5.377$, $p < 0.05$). Therefore, participants who tried various options by trial and

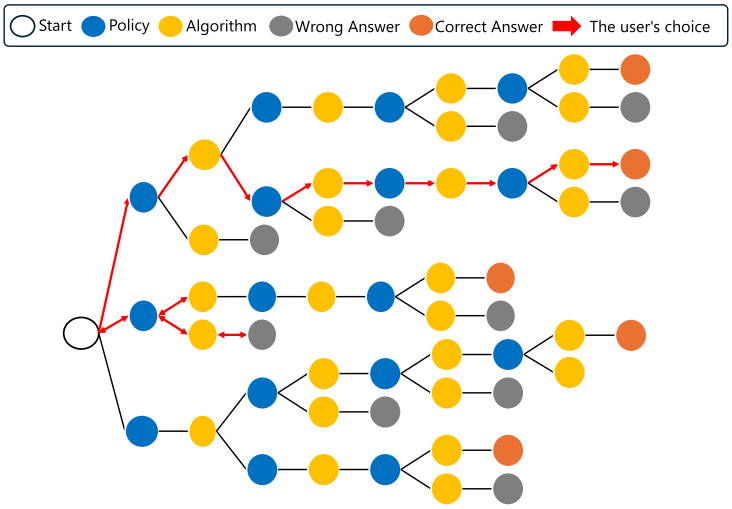

FIGURE 11. Example of exploring the options (Task 2-1).

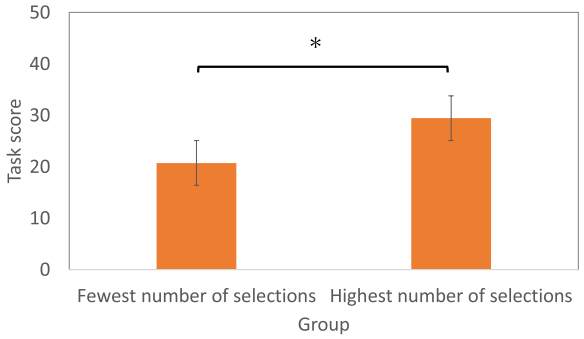

FIGURE 12. Comparison of class task scores.

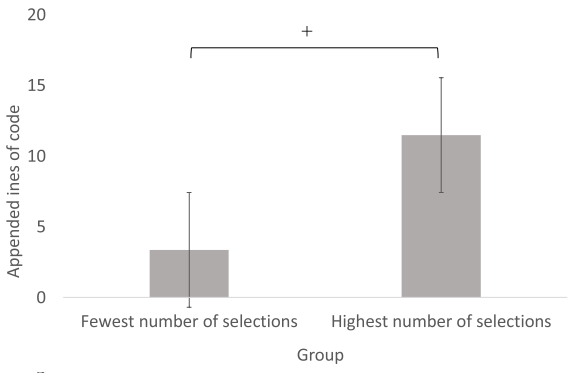

FIGURE 13. Comparison of lines of code.

error on the proposed system may improve their skills in solving programming tasks.

Although there was no correlation between class task scores and the number of times participants backtracked, the correlation with the number of participant choices is likely because some participants solved the same task multiple times, resulting in a high system usage rate.

B. SUBMITTED PROGRAMS

We analyzed the codes submitted by the participants in a class task. There was a significant correlation between the number of choices the proposed system was selected and the number of lines of code submitted by the participants ($r = 0.412$, $p = 0.035$). We grouped the participants according to the total number of choices they made on the proposed system. We also compared the class task scores of the top 25% group (seven participants) and the bottom 25% group (seven participants) regarding the number of choices made using a one-factor within-participant analysis of variance. The results are shown in Figure 13, and there was a significant trend in the number of lines of code between groups ($F_{(1,12)} = 4.628$, $p < 0.1$).

Therefore, users who use the proposed system more frequently can write more code. While this indicates an improvement in users' problem-solving skills in programming, it also shows a possible decrease in the efficiency and readability of the written code.

C. QUESTIONNAIRE RESULT

We present the results of the proposed system questionnaire. 37 participants (17 males and 20 females) responded to the questionnaire. The results of the questionnaire are shown in Table 5.

First, the questions about solution planning (Q1-Q4) were highly rated by the participants, indicating that the proposed system may have been able to support solution planning. In the free description, some comments appreciated the skill to plan solutions in natural language, such as "I could understand the meaning in Japanese" and "It was easy to understand because I could think of the policy in Japanese sentences, not in program code." In addition, comments such as "I was able to simulate the thinking of a person writing a program" and "I was able to experience the order

TABLE 5. Questionnaire Result (1: totally disagree - 5: strongly agree).

Code	Question	AVE	SD
Q1	Do you feel that your ability to develop a strategy for solving problems has improved?	4.03	0.87
Q2	Could you concentrate more on planning a strategy for solving the task than writing the program directly?	3.92	1.01
Q3	Was it easier to plan a strategy than to write the program directly?	4.14	1.00
Q4	Were you able to think about solutions step by step?	4.16	0.99
Q5	Were you able to try out different options easily?	4.03	1.12
Q6	Did you do a quick trial-and-error to solve the problem?	4.03	1.01
Q7	Was it easier to do the tasks in class?	4.24	1.04
Q8	Was the application easy to use?	3.76	1.09

in which programs are created'' indicated that simulating programmers' thinking processes was helpful in planning solutions.

Moreover, the proposed system supports problem-solving by eliminating the coding process, allowing novice users to focus on planning solutions. Comments on this feature included, ''Because I did not have to spend so much time debugging, I was able to focus on thinking about the policy without getting tired.'' and ''I did not have to look up specific code.'' Therefore, eliminating the coding process and allowing the user to focus on the solution can help the novice learner plan a solution.

The high rating of the question (Q5-Q7) whether the system could realize easy and fast trial and error suggests that the proposed system may realize easy and fast trial and error.

On the other hand, the evaluation of the usability of the proposed system in Q8 was divided into the following opinions, and points for improvement of the proposed system were found:

- ''It was good that options were presented.''
- ''It was easy to use and understand.''
- ''Because it was multiple choice, it was easy to think in an organized way.''
- ''Sometimes clicking the button did not do anything.''
- ''There were times when screen transitions felt a bit slow.''
- ''I tend to forget to enter my username every time, so I'd like it to be remembered even when I leave the site.''

Participants commented on the use of the proposed system as follows:

- ''It was good that I could easily prepare for the class.''
- ''I would like to use this system in future classes because I think it has the best elements as a preparation material.''
- ''I would like to use this system again in the four quarters.''
- ''I can easily touch programming and think in Japanese, so it was easy to use.''
- ''It was fun, and I would like to be involved in programming in the future. I would like to use it again in the future.''

These comments indicate high satisfaction with the proposed system and suggest that it may create a positive impression about learning programming.

VII. DISCUSSION

A. LIMITATION

The questionnaire survey results indicate that the participants positively evaluated the system's functionality for supporting solution planning. However, the proposed system's usability was hindered by issues such as poor response to clicks, delays in page transitions, and the need to enter the user's name each time. These issues may also hinder users' problem-solving and trial-and-error processes and, therefore, require improvement in the future.

In the empirical study, we applied the proposed system in an introductory programming course for university students in the humanities. If the prerequisite knowledge of the target users is different, for example, elementary school students with insufficient knowledge of English vocabulary used in programming and engineering students with some knowledge of computers, the proposed system may only be partially effective. As noted in Section V.A, the participants in the empirical study worked on the tasks in groups so that the results may include the influence of group work.

B. FUTURE WORK

1) GENERATING OPTIONS FOR PROGRAM TASKS

In this paper, the first author manually prepared the solution options for program tasks. However, it can be difficult for a teacher to manually prepare choices for each task in an educational setting like a lecture or workshop. We will implement a function to quickly generate options for prepared program tasks to address this issue. The options for program tasks can be simplified by using a graphical editor that expresses paths for answers as nodes or an LLM like ChatGPT to generate options that the teacher can edit.

2) APPLICATION TO OTHER PROGRAMMING AREAS

The results of the empirical experiment show that the proposed system is stable and can support learning in actual classroom use. The implemented system in the empirical experiment aims to facilitate programming learning with visual output through p5.js. While the system's UI is specific to p5.js, it can be applied to programming learning in other subjects and using different programming languages. For instance, the system could be used to learn programming languages other than JavaScript. Technologies such as WebAssembly [38] can convert programs written in Python,

Java, and other languages into a browser-executable format, enabling similar learning experiences in different languages. Other potential applications of programming education with content include learning to design a program that takes data as input and returns a specific output or handling software development, such as games or web development.

3) TOWARD PROGRAMMING WITH AI SUPPORT TOOLS

Programming styles are evolving with the increasing use of AI-based coding support tools. LLMs like ChatGPT enable non-programmers to create programs based on requirements described in natural language. This means that even non-programmers need to understand programming and be able to define software requirements. The proposed system will serve as an introduction for non-programmers who need an understanding of programming.

VIII. CONCLUSION

In this study, we proposed a programming learning support system that presents a problem-solving strategy for programming in natural language. The system aims to enhance the problem-solving skills of novice programmers by allowing them to consider solutions from a limited set of alternatives through trial and error.

We implemented the proposed system as a web application and used it in an introductory programming course for beginners at a university. The study discovered a correlation between the frequency of student choices made using the proposed system and their class task scores. This suggests that the system may improve problem-solving skills for programming tasks. Additionally, the study found a correlation between the frequency of user choices in the proposed system and the number of lines of code used to complete the class task. Users positively evaluated the questionnaire survey results regarding the proposed system.

Based on the results of our experiments, we will improve the system's response and other areas for improvement. Additionally, we plan to expand the system's application to other programming fields and add the capability to generate task options.

REFERENCES

[1] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 1, pp. 33–35, Mar. 2006.

[2] D. Weintrop, E. Beheshti, M. Horn, K. Orton, K. Jona, L. Trouille, and U. Wilensky, "Defining computational thinking for mathematics and science classrooms," *J. Sci. Educ. Technol.*, vol. 25, no. 1, pp. 127–147, Feb. 2016.

[3] *Code.org*. Accessed: Apr. 10, 2024. [Online]. Available: https://code.org/

[4] *Codecademy*. Accessed: Apr. 10, 2024. [Online]. Available: https://www.codecademy.com/

[5] *CodeCrafters*. Accessed: Apr. 10, 2024. [Online]. Available: https://codecrafters.io/

[6] H. Lieberman, F. Patern, M. Klann, and V. Wulf, "End-user development: An emerging paradigm," in *End User Development*. Dordrecht, The Netherlands: Springer, Jan. 2006, pp. 1–8.

[7] S. Lau, S. S. Ragavan, K. Milne, T. Barik, and A. Sarkar, "Tweakit: Supporting end-user programmers who transmogrify code," in *Proc. CHI*, Yokohama, Japan, 2021, pp. 1–12.

[8] *ChatGPT*. Accessed: Apr. 10, 2024. [Online]. Available: https://chat.openai.com/

[9] S. Biswas, "Role of ChatGPT in computer programming," *J. Comput. Sci.*, vol. 2023, pp. 8–16, Mar. 2023.

[10] S. R. M. Derus and A. Z. M. Ali, "Difficulties in learning programming: Views of students," in *Proc. ICCIE*, Yogyakarta, Indonesia, 2012, pp. 74–78.

[11] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," in *Proc. ITiCSE*, Caparica, Portugal, 2005, pp. 14–18.

[12] M. N. Ismail, N. A. Ngah, and I. N. Umar, "Instructional strategy in the teaching of computer programming: A need assessment analyses," *Turk. Online J. Educ. Technol.*, vol. 9, no. 1, pp. 125–131, 2010.

[13] G. Bain and I. Barnes, "Why is programming so hard to learn?" in *Proc. ITiCSE*, 2014, p. 356.

[14] R. Lister, "Toward a developmental epistemology of computer programming," in *Proc. WiPSCE*, New York, NY, USA, 2006, pp. 5–16.

[15] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Proc. IEEE Symp. Vis. Lang.-Hum. Centric Comput.*, Sep. 2004, pp. 199–206.

[16] E. Soloway, "Learning to program = learning to construct mechanisms and explanations," *Commun. ACM*, vol. 29, no. 9, pp. 850–858, Sep. 1986.

[17] A. N. Kumar, "Learning programming by solving problems," in *Informatics Curricula and Teaching Methods*. Boston, MA, USA: Springer, 2003, pp. 29–39.

[18] J. Shi, A. Shah, G. Hedman, and E. O'Rourke, "Pyrus: Designing a collaborative programming game to promote problem solving behaviors," in *Proc. CHI*, Glasgow, U.K., 2019, pp. 1–12.

[19] *Processing*. Accessed: Feb. 18, 2024. [Online]. Available: https://processing.org/

[20] *P5.js*. Accessed: Apr. 10, 2024. [Online]. Available: https://p5js.org/

[21] M. Resnick, J. Maloney, A. Monroy-Hernndez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 1, pp. 60–67, 2000.

[22] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic books, 2020.

[23] C. Kelleher, R. Pausch, and S. Kiesler, "Storytelling Alice motivates middle school girls to learn computer programming," in *Proc. CHI*, 2007, pp. 1455–1464.

[24] D. Halbert. *Programming by Example*. Accessed: Feb. 18, 2024. [Online]. Available: https://danhalbert.org/pbe-html.htm

[25] J. Kato and M. Goto, "Live tuning: Expanding live programming benefits to non-programmers," in *Proc. LIVE*, Rome, Italy, 2016, pp. 1–6.

[26] K. Krings, N. S. Bohn, N. A. L. Hille, and T. Ludwig, "'What if everyone is able to program?'—Exploring the role of software development in science fiction," in *Proc. CHI*, Hamburg, Germany, 2023, pp. 1–13.

[27] J. Kato and K. Shimakage, "Rethinking programming 'environment': Technical and social environment design toward convivial computing," in *Proc. Program*, New York, NY, USA, 2020, pp. 149–157.

[28] P. Brusilovsky, A. Kouchnirenko, P. Miller, and I. Tomek, "Teaching programming to novices: A review of approaches and tools," in *Proc. EdMedia*, Vancouver, BC, Canada, 1994, pp. 103–110.

[29] R. B. Findler, C. Flanagan, M. Flatt, S. Krishnamurthi, and M. Felleisen, "DrScheme: A pedagogic programming environment for scheme," in *Proc. PLILP*, Southampton, U.K., 1997, pp. 369–388.

[30] E. Roberts, "An overview of MiniJava," in *Proc. SIGCSE*, Charlotte, NC, USA, 2001, pp. 1–5.

[31] F. Hermans, "Hedy: A gradual language for programming education," in *Proc. ICER*, Aug. 2020, pp. 259–270.

[32] G. Dietz, J. K. Le, N. Tamer, J. Han, H. Gweon, E. L. Murnane, and J. A. Landay, "StoryCoder: Teaching computational thinking concepts through storytelling in a voice-guided app for children," in *Proc. CHI*, Yokohama, Japan, May 2021, pp. 1–15.

[33] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, problem solving, and self-awareness: Effects of explicit guidance," in *Proc. CHI*, San Jose, CA, USA, May 2016, pp. 1449–1461.

[34] D. W. Sandberg, "Smalltalk and exploratory programming," *ACM SIGPLAN Notices*, vol. 23, no. 10, pp. 85–92, Oct. 1988.

[35] M. B. Kery and B. A. Myers, "Exploring exploratory programming," in *Proc. VL/HCC*, Raleigh, NC, USA, 2017, pp. 25–29.

[36] T. Angert, M. Suzara, J. Han, C. Pondoc, and H. Subramonyam, "Spellburst: A node-based interface for exploratory creative coding with natural language prompts," in *Proc. UIST*, 2023, pp. 1–22.

[37] *Next.js*. Accessed: Apr. 10, 2024, [Online]. Available: https://nextjs.org/

[38] *WebAssembly*. Accessed: Apr. 10, 2024. [Online]. Available: https://webassembly.org/

HIROKI OKA received the B.E. and M.A. degrees from Kobe University, Hyogo, Japan, in 2019 and 2021, respectively, where he is currently pursuing the Ph.D. degree with the Graduate School of Engineering. His research interests include programming education, wearable computing, and ubiquitous computing.

AYUMI OHNISHI received the B.Eng. degree from Kobe University, in 2014, the M.Env. degree from The University of Tokyo, in 2016, and the Ph.D. degree from Kobe University. She is currently an Assistant Professor with the Graduate School of Engineering, Kobe University. She is working on wearable computing and ubiquitous computing technologies.

TAKESHI NISHIDA received the B.S. degree in information science and the M.S. and Ph.D. degrees in information science and technology from The University of Tokyo, in 2004, 2006, and 2009, respectively. From 2010 to 2012, he was a Lecturer with the Graduate School of Intercultural Studies, Kobe University, Hyogo, Japan, where he has been an Associate Professor, since 2013. His research interests include universal design in communication, focusing on creating inclusive communication systems that accommodate diverse personalities and situations.

TSUTOMU TERADA (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in engineering from Osaka University, Osaka, Japan, in 1997, 1999, and 2003, respectively. From 2000 to 2004, he was an Assistant Professor with the Cybermedia Center, Osaka University, where he was a Lecturer, from 2005 to 2007. From 2007 to 2018, he was an Associate Professor with the Graduate School of Engineering, Kobe University, Hyogo, Japan, where he has been a Professor, since 2018. His research interests include wearable computing, ubiquitous computing, and entertainment computing.

MASAHIKO TSUKAMOTO (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in engineering from Kyoto University, Kyoto, Japan, in 1987, 1989, and 1994, respectively. From 1995 to 1996, he was a Lecturer with the Graduate School of Engineering, Osaka University, Osaka, Japan, where he was an Associate Professor, from 1996 to 2004. Since 2004, he has been a Professor with the Graduate School of Engineering, Kobe University, Hyogo, Japan. His research interests include wearable computing and ubiquitous computing.