Yamashita et al. *Research and Practice in Technology Enhanced Learning* (2023) 18:33
https://doi.org/10.58459/rptel.2023.18033

Research and Practice in Technology Enhanced Learning

RESEARCH Free and Open Access

# Interaction support systems between teachers and visual content for effortless creation of program visualization

Koichi Yamashita [1] *, Miyu Suzuki [2], Yusuke Kito [2], Yusuke Suzuki [2], Satoru Kogure [2], Yasuhiro Noguchi [2], Raiya Yamamoto [3], Tatsuhiro Konishi [2] and Yukihiro Itoh [4]

*Correspondence:
yamasita@hm.tokoha-u.ac.jp
[1] Faculty of Business Administration,
Tokoha University,
1230 Miyakoda,
Kita-ku, Hamamatsu, Shizuoka
431-2102, Japan
Full list of author information is available at the end of the article

**Abstract**

Several program visualization (PV) systems have been developed to support novice learners in understanding program behavior since last couple of decades. However, only a few have been introduced or continuously used in actual classes. One of the main obstacles to using PV systems in actual classrooms is the significant amount of time needed to integrate them into actual educational settings. We developed a PV system called Teacher's Explaining Design Visualization Tool (TEDViT) and introduced it into several practical applications. Although programming learning with TEDViT had a noticeable effect, the time required for PV customization (i.e., the time consumed for interactions between teachers and PV content) was a non-trivial problem. In this study, we describe three approaches to reduce the time cost of customizing by teachers; that is, we supported PV creation by (1) semi-automatically arranging drawing objects oriented toward novice learners, (2) allowing menu operations with a dialog interface, and (3) providing visual information and visual operations using a WYSIWYG PV editor. We developed three individual systems based on each approach and evaluated their effortlessness by measuring the time required for actual PV creation. The evaluation results suggest that each of the three approaches has a certain effect on improving the effortlessness of PV creation. This study describes our three approaches and the system developed based on them and discusses the possibility of integrating them.

**Keywords:** Programming education, Program visualization system, Program visualization design, Educational authoring tool

## Introduction

Program visualization (PV) is a widely accepted approach for supporting novice learners who struggle to clearly understand program behavior. Several PV systems have been developed, and many positive learning effects have been reported (Pears et al., 2007).

However, few PV systems have been continuously introduced in actual classes. One of the main obstacles to their continuous use is the time cost involved. Teachers who introduce PV systems into their classrooms must design, integrate, and maintain the PV generated by the system alongside their lesson plans.

To address this issue, we developed a PV system, the Teacher's Explaining Design Visualization Tool (TEDViT), and conducted several classroom practice sessions using it (Kogure et al., 2014). One distinctive feature of TEDViT is that it enables teachers to customize PVs based on their own instruction plans. Through this feature, teachers can design, integrate, and maintain PVs that reflect their intentions, thereby achieving positive evaluation results that suggest significant learning effects (Yamashita et al., 2016; Yamashita et al., 2017). However, compared to existing PV systems, this feature incurs additional costs for PV customization. Further research is required to reduce the cost of using PVs.

While many factors may account for the high cost of PV creation, this study focused on PV customization, i.e., interactions between teachers and PV content, the most direct way to increase the learning effectiveness of PV systems. In this study, we describe three approaches for improving the effortlessness of PV creation: (1) support interaction by automatically arranging drawing objects oriented toward novice learners, (2) support it by menu operations with a dialog interface, and (3) support it visually with the WYSIWYG PV editor. We define the research questions in this study as follows:

RQ1. To what extent can each of the three approaches developed to support PV customization reduce costs?

RQ2. How can each of these three approaches be applied to reduce costs?

We developed three individual systems based on each approach and evaluated their effortlessness by measuring the time required for actual PV creation. The evaluation results suggest that each of the three approaches has a certain effect on improving the effortlessness of PV creation.

## Related studies

### Existing PV systems

Over the past few decades, several PV systems have been developed for novice learners, including Python Tutor (Guo, 2013), Jype (Helminen & Malmi, 2010), and PROVIT (Yan et al., 2014). These systems differ in several ways. For example, Python Tutor runs on a web browser and does not require local installation. Jype provides a learning environment that integrates the PV and automatic assessment systems for exercise assignments. PROVIT uses 3D graphics for visualization. However, all these systems are similar in that

they visualize the target program and its data structures in a uniform manner. Generally, these systems can visualize programs using a fixed visualization policy. They also allow learners to observe changes in data structure during the execution of each statement. This function is provided by a graphical user interface (GUI), such as next/previous buttons, for stepwise execution of the target program. Sorva et al. (2013) provided a comprehensive overview of more than 40 PV systems that share several similarities.

PV systems demonstrate the runtime behavior of computer programs for novice learners by visually encoding data and showing how it is processed in a running program. Novice learners often find it difficult to trace program states and behaviors by using data structures. By bridging the gap between their reasoning and computational processes, PV systems can improve novice learners' understanding of programs (Tudoreanu, 2003).

However, as Sirkiä and Sorva (2015) pointed out, PV systems are not always effective. Learners may struggle to understand the meaning of visual elements, neglect important aspects, or focus on peripheral elements. We would argue that this reflects a failure to integrate with other materials or offer customizable systems. This also suggests a poor fit with the teachers' personal pedagogical styles. Sorva et al. (2013) call this the "problem of dissemination." In this study, teachers we are referring to are programming teachers. While learners' reasoning is formed by the teacher's in-class explanations, computational processes are visualized based on the designs of PV system developers. Hence, there is a gap between teachers' explanations and their PVs.

For example, a teacher might draw an array object in a horizontal layout when the instruction target is to sort an array, whereas the teacher might draw an array in the vertical layout for a stack. Changes in visualization policies, such as these, are derived by fitting the instruction content to the learners' background knowledge. If learners sufficiently understand a stack, drawing either an object in a horizontal or vertical layout would be acceptable to them. Similarly, the teacher would not need to draw the temporary variable in a task that swaps the values of the two variables for non-novice learners. We call teachers' responses to individual learning situations as intent of instruction.

The simplest way for teachers to provide learners with a visual understanding of the computational process by incorporating their own intent of instruction is to use presentation software, such as PowerPoint. However, because this method can only offer PVs fixed to a specific processing target data, teachers and learners cannot flexibly change the target data on-the-fly to explain changes in the computational process and observe changes, respectively. We believe that a PV system that can customize PV based on the teacher's intent of instruction is required to solve these problems.

A few systems, such as Jsvee & Kelmu (Sirkiä, 2018) and ANIMAL (Rössling & Freisleben, 2002), can customize PVs. Jsvee & Kelmu is PV system consisting of two subsystems: Jsvee, which can automatically create PVs similar to many other existing PV

systems, and Kelmu, which allows teachers to annotate the PVs generated by Jsvee. Kelmu can generate annotations by text to explain program behaviors and quizzes that require learners to answer questions. As Naps et al. (2002) and Ihantola et al. (2005) pointed out, interaction between PVs and learners is a promising approach for improving learners' engagement. However, Jsvee & Kelmu does not allow teachers to customize the drawing objects generated by Jsvee based on their intents of instruction. ANIMAL is a system that allows teachers to freely define PVs from scratch using a script called AnimalScript. Generally, there is a trade-off between PV customizability and the cost of customization. The high degree of freedom in defining PVs using ANIMAL requires a great deal of prior knowledge and preparation for PV customization. The cost of learning AnimalScript is significant, and PV creation requires a non-trivial quantity of script code. The sample script for the bubble-sort algorithm bundled in ANIMAL consisted of 170 lines of script code. To address these issues, efforts to reduce the cost of PV creation are required.

## Effortless PV creation

Several studies have investigated methods to reduce the cost of algorithm visualization (AV) and PV creation. AV systems visualize general algorithms, whereas PV systems visualize the execution processes of concrete programs. Sorva et al. (2013) pointed out that the difference between AV and PV systems is their levels of abstraction and that PV systems tend to provide visualizations at a lower level than AV systems.

Malone et al. (2009) developed a pseudocode system in which the definition of visualization can be included in the pseudocode used to represent the target algorithm. Because the pseudocode interpreter automatically derives the AV from the algorithm implementations, they argue that the effectiveness and effortlessness of AV improved by their system. Velázquez-Iturbide et al. (2008) developed a system that allows teachers to select PVs from an automatically generated PV sequence in list format. Although they argued that their system improves effortlessness in PV creation, they did not present any experimental results to prove this objective. Rößling and Ackermann (2007) developed a framework that allows teachers to create AV content on-the-fly by adjusting variable values and attributes. Their framework was derived from a set of prepared templates that defined the visualization details of various algorithms. These definitions can be saved freely, thereby reducing the effort required to reuse the AVs. PV systems such as Jeliot 3 (Moreno et al., 2004) are often considered effortless because they automatically generate PVs by providing target programs only, although the PVs generated in this manner cannot be customized. There are various approaches to effortless PV creation, but simple comparisons are difficult.

Ihantola et al. (2005) defined a taxonomy to characterize effortlessness in AV systems. Based on a survey of computer science (CS) educators, they identified three main

categories—*scope*, *integrability*, and *interaction*—and evaluated several existing systems. The *scope* refers to the range of contexts in which the AV system can be applied, i.e., the various algorithmic domains for which the system can be adapted. *Integrability* refers to third-party effortlessness: how easy it is to integrate the AV system into educational setups. *Interaction* refers to the extent to which a system can be used in various cases. This factor is based not only on interactions between AV content and learners but also on interactions between teachers and content and the extent to which the content is customizable.

These factors are applicable not only to AV creation but also to PV creation. Because one of the goals of PV systems is to help users understand the underlying algorithms by visualizing program behavior, we applied the three perspectives on effortless AV creation directly in the context of PV creation. While it is necessary to add new and different factors (e.g., supporting programming languages, language-specific features, etc.) in the context of PV creation, this study focuses on the cost of customizing the visualizations of program execution processes. In other words, we focused on the interaction between teachers and visual content.

The reasons for focusing on this are mainly based on our previous classroom practices using TEDViT (Ihara et al., 2017; Kogure et al., 2014; Kogure et al., 2018; Yamamoto et al., 2017; Yamashita et al., 2016; Yamashita et al., 2017; Yamashita et al., 2020). TEDViT is a PV system that allows teachers to customize PV based on their own intents of instructions. The practice classes obtained positive learning effects from appropriate interactions between teachers and content (i.e., PV customization). We observed that customized PVs cultivated learners' better understanding of programs (Yamashita et al. 2017), allowing learners to use PV systems as a tool for discovery learning (Yamashita et al. 2016), and so on. Hence, the goal of this study was to improve the effortlessness of PV creation by developing a system that supports teachers' PV customization.

## TEDViT

The TEDViT system interprets each visualization policy by scanning the configuration file and visualizing PVs accordingly. Teachers can create PVs based on their intents of instruction by providing TEDViT with their configuration file in addition to the target program. Figure 1 shows a screenshot of the learning environment visualized using TEDViT. The configuration file comprises a set of drawing rules, each of which is a comma-separated value (CSV) entry consisting of a condition and an object. This condition defines the prerequisites required for firing the drawing rule. Teachers can use a conditional equation (consisting of a statement ID, variables in the target program, constant values, and comparison operators) to determine drawing timing. Here, the statement ID is a unique identifier automatically assigned to all the statements in the target program using TEDViT. The object defines the operation ("create," "delete," or "update") used to edit the target

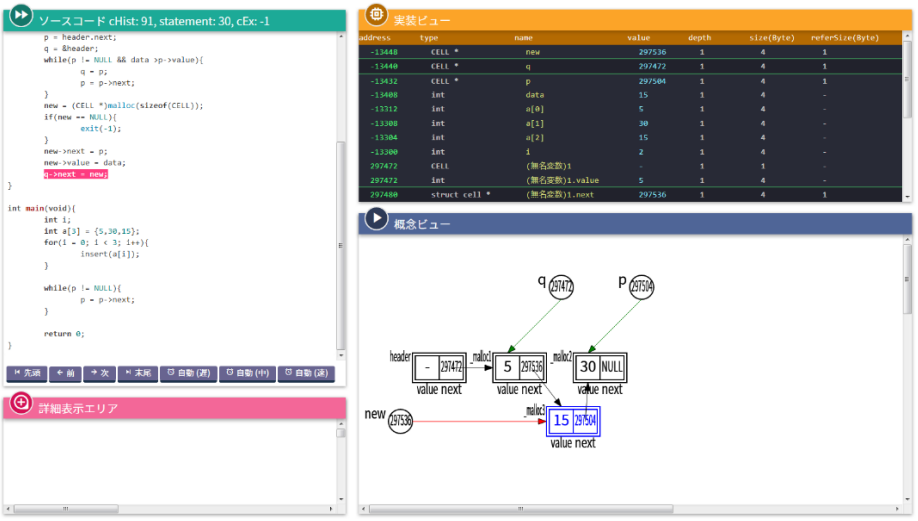


**Fig. 1** Screenshot of a learning environment generated by TEDViT

object and the attributes necessary for drawing it, which include object type, position, color, and corresponding variables.

TEDViT generates execution history by actually executing the target program, which is used to judge the conditions for firing drawing rules and reading the values of drawing objects. The process of generating the execution history is illustrated in Figure 2. Currently, TEDViT supports only the visualization of C programs. Because the framework for generating execution histories is relatively simple, almost all functions that novice learners would learn are supported in the visualization by TEDViT. Several programs can be visualized, including the behavior of functions with recursive calls (Yamamoto et al., 2017), dynamic data structures (Yamashita et al., 2019) and pointer behavior (Yamashita et al., 2020). On-the-fly visualization is also possible because the execution history is independent of the PV definitions, and the execution history can be easily updated if the input data to the program are changed (Yamashita et al., 2016). However, it cannot be used as a debugger for programs that contain compilation errors because execution history cannot be generated from programs that cannot be executed.

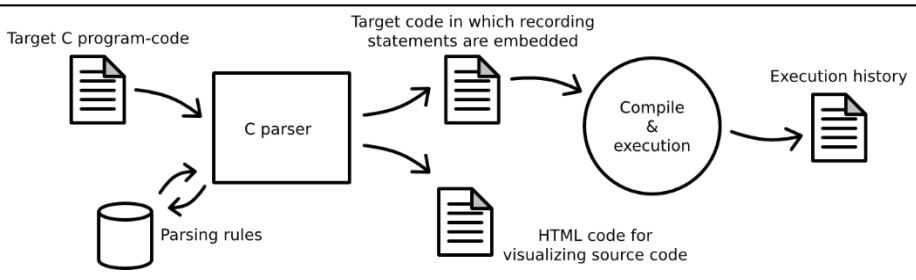


**Fig. 2** Process of generating execution history in TEDViT

```
state==10, create, OBJ1, circle, new, x1, y6, black, white, black, , , solid, arrow, , pointer, red
```

**Fig. 3** Example of a TEDViT drawing rule

Figure 3 shows an example of this drawing rule. This implies that when the statement with ID "10" in the target program is executed, TEDViT draws a circle object and assigns it the object ID "OBJ1." The corresponding variable is the pointer variable "*new*"; hence, the value of *new* (the address of the variable it refers to) is drawn inside OBJ1. OBJ1 is placed at position (x1, y6) with black, white, and black as the line, background, and inner character colors, respectively, in accordance with the values indicated in the rule. Moreover, the pointer reference is visualized with red and solid arrows (see Figure 1). We consider that setting attributes such as object shapes, colors, and line thicknesses does not only mean drawing adjustments, but is also performed by the teacher as emphatic expressions to provide the focusing points in the PVs to the learners.

The customizability of PV with TEDViT is provided by a high degree of freedom to define these drawing rules. Although the customizability of PVs based on drawing rules is somewhat limited compared to the customizability of ANIMAL, which allows teachers to freely define PVs from full scratch, the number of definitions required to generate PVs is smaller than that in AnimalScript. Yamashita et al. (2016) reported that TEDViT requires only 56 lines of drawing rule definitions to generate PVs comparable to the bubble-sort PVs bundled in ANIMAL, which is defined as 170 lines of AnimalScript. TEDViT provides buttons for the stepwise control of the target program execution, similar to the GUI in typical PV systems. When a learner clicks on the "previous" and "next" buttons, TEDViT finds the corresponding program-execution status, fires the rule for which the condition is satisfied, and visualizes the corresponding drawn objects.

We conducted several classroom practices sessions incorporating TEDViT into actual classes to evaluate its learning effects (Kogure et al., 2018; Yamashita et al., 2016; Yamashita et al., 2017; Yamashita et al., 2020). To allow teachers to reflect on their intents of instruction in their PVs, our targeting use style of TEDViT was the same as in previous practice sessions, where the teacher provides it to the learners as a learning environment to explain program code during classroom exercises. In classroom practice sessions with TEDViT, the use of PVs that reflected the teacher's intent of instruction led to positive learning effects. However, the customizability of PV in TEDViT also creates a burden for teachers, who must define drawing rules. According to Yamashita et al. (2016), it took approximately 30 min to define the drawing rules for a single-sample selection-sort code consisting of approximately 30 statements. Although teachers rated this as an acceptable class preparation cost, we considered it a significant burden.

TEDViT is a browser-based PV system that does not require separate installations. Moreover, as it can provide PVs in accordance with teachers' intents of instruction, it is considered to have a sufficient level of *integrability* on the effortlessness discussed by Ihantola et al. (2005). In addition, because TEDViT supports most of the grammatical targets learned by novice learners of C programming, it can be evaluated as a system with a wide *scope*. Some studies have shown that TEDViT supports object-oriented languages such as Java (Kogure et al., 2019); hence, TEDViT can be regarded as not only a course-specific PV system but also a domain-specific system. However, TEDViT does not provide sufficient support for *interaction*, which is the focus of this study. Therefore, by adopting TEDViT as a PV system, effortless PV creation can be achieved if the cost of interactions is reduced by supporting them.

Ihantola et al. (2005) call an interaction between AV/PV and learners a visualization-consumer interaction (VC interaction) and an interaction between the AV/PV system and teachers a producer-system interaction (PS interaction). While we have also extended TEDViT to include features of VC interaction, such as automatic assessment for self-study (Kogure et al., 2018), the goal of this study is to help teachers to reflect their intents of instruction in their PVs. Therefore, in this study, effortlessness for PV creation was considered only from the perspective of PS interaction, and VC interaction was not considered. This study was an attempt to improve the effortlessness of PV creation by adopting TEDViT as a PV system and reducing the cost of PS interaction.

## Effortless PV creation based on semi-automatic arrangement of drawing objects

### Approach

The editing of CSV files in TEDViT brings a high degree of freedom in PV definition to teachers (i.e., PV creators), but this high degree of freedom places excessive cognitive loads on teachers, which is one of the reasons for the cost of PV creation. However, the importance of a high degree of freedom can decrease if the system is limited to programming education for novice learners. The visual representations used to explain data structures in several textbooks for novice programmers are similar, and teachers' explanations are considered to be influenced by those representations. For example, the visual representation of an array structure is almost always a sequence of adjacent squares with internal values regardless of whether the layout is vertical or horizontal. In many cases, the visual representations that teachers use to explain array structures in the classroom are similar.

If visual representations for novice learners have a certain similarity, an approach in which the system supports PV creation by suggesting drawing objects as typical

visualizations for novice learners could be considered. The system can have a certain level of customizability by providing a dialog interface to input some parameters, such as the coordinates of the drawing objects. Although visualization based on a dialog interface is a widely accepted approach (as seen in Atemezing and Troncy (2014) and Roth et al. (2013), etc.), it is mostly used to visualize some data and has not been applied in many cases to targets with dynamics, such as PV. TEDViT generates PVs based on a configuration file consisting of a set of drawing rules. This approach can be used to generate drawing rules and does not directly generate visualization content. Hence, in the case of TEDViT, effortlessness can be improved without a loss of customizability by editing the PV definitions generated by the system. Because TEDViT visualizes PVs according to a ruleset, this approach generates drawing rules to visualize all data structures declared in the target program as typical PV objects. Because novice learners are the main users of PV systems, it is expected that the cost of PV creation could be significantly reduced compared with creating rulesets by editing CSV files from scratch.

Based on this consideration, we developed a system that generates rules to draw variables and arrays declared in the target program in a uniform manner, and implemented a dialog interface to input parameters for drawing PV objects. Figure 4 shows a screenshot of the dialog interface. Our system requires the input of a file “history.js” instead of the target program code, which is the execution history file automatically outputted by TEDViT when parsing the target code to generate PV (see Figure 2).

When “history.js” is entered, the interface changes from Figure 4 to Figure 5, showing available parameter forms. In this interface, input forms are provided to specify the coordinates of each drawing object and to determine whether the variable is used as an index for the array element. The indexer form was implemented because indexers are PV elements that are frequently used in classroom practices using TEDViT. All the input forms

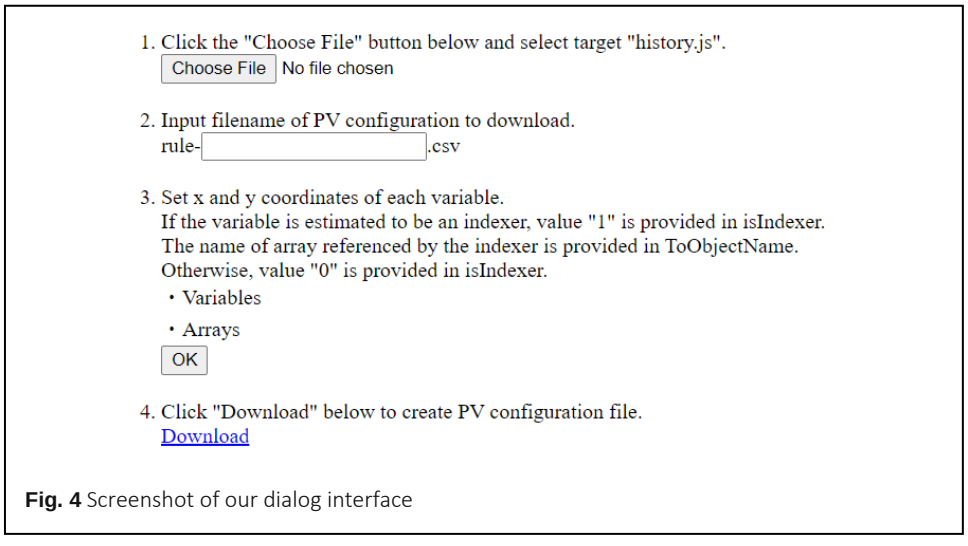


**Fig. 4** Screenshot of our dialog interface

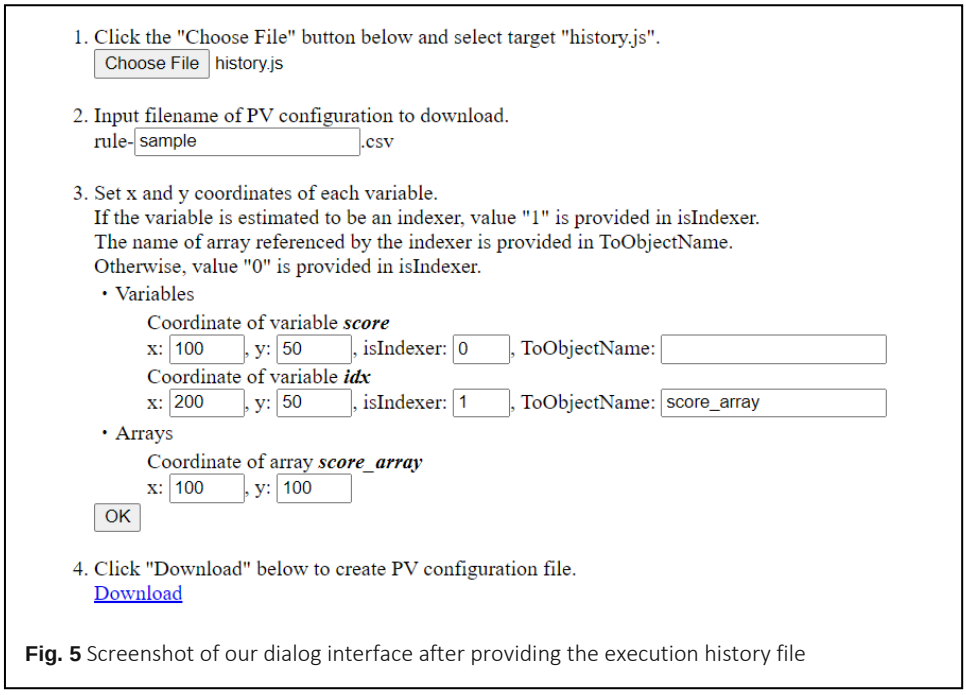


**Fig. 5** Screenshot of our dialog interface after providing the execution history file

for the coordinates of the drawing object and whether it is an indexer are given initial values automatically estimated by our system, and the teachers update the values only when it is necessary to change them. After specifying parameters, the ruleset file in CSV format can be obtained by clicking the “Download” link.

## Evaluation experiment

To evaluate the effectiveness of PV creation using our system, we conducted an experiment to measure the actual time required to create PVs. A survey by Naps et al. (2002) found that more than 90 percent of participants at the ITiCSE 2002 conference cited the time required for PV creation as a factor in their reluctance to use animation (i.e., PV sequences). Thus, the evaluation of effortlessness based on the time required for PV creation was considered valid.

Twelve participants were involved in this experiment: two teachers with over 10 years of experience in teaching programming at university, two 23-years-old CS master’s students with experience as programming teaching assistants, and eight 22-years-old undergraduate students with the same level of programming experience as the teaching assistants. We prepared three sample programs and sample PVs as PV creation targets: linear-search, binary-search, and bubble-sort programs. Each participant received two sample programs and PVs, and was asked to create the same PVs as the sample PVs. We asked the participants to create PVs with and without our system for each sample program; hence, the participants performed the task four times. To reduce the order effects, we first specified

whether each participant would use our system. Table 1 summarizes the conditions of each participant. In Table 1, items with "+" at the end of the target program indicate the tasks using our system. Figure 6 shows the sample program we prepared for bubble-sorting, and Figure 7 shows a sample PV. We also prepared sample sets of similar complexity for linear- and binary-searches; however, we omit them here.

We began this experiment by explaining to the participants (for 15 min) the specification of the TEDViT drawing rules and how to use our system. Next, we gave them two sample programs and sample PVs without disclosing the drawing rules and asked them to "define drawing rules to reproduce the same PV as the provided sample PV." We controlled the participants' use of the system by asking "use (or do not use) our system to perform the

**Table 1** Conditions per each participant ("+" means task with our system)

| Participants # | 1st task | 2nd task | 3rd task | 4th task |
|---|---|---|---|---|
| 1 | Linear-search | Bubble-sort | Linear-search+ | Bubble-sort+ |
| 2 | Bubble-sort | Linear-search | Bubble-sort+ | Linear-search+ |
| 3 | Binary-search | Bubble-sort | Binary-search+ | Bubble-sort+ |
| 4 | Bubble-sort | Binary-search | Bubble-sort+ | Binary-search+ |
| 5 | Bubble-sort | Linear-search | Bubble-sort+ | Linear-search+ |
| 6 | Binary-search | Bubble-sort | Binary-search+ | Bubble-sort+ |
| 7 | Linear-search+ | Bubble-sort+ | Linear-search | Bubble-sort |
| 8 | Bubble-sort+ | Linear-search+ | Bubble-sort | Linear-search |
| 9 | Binary-search+ | Bubble-sort+ | Binary-search | Bubble-sort |
| 10 | Bubble-sort+ | Binary-search+ | Bubble-sort | Binary-search |
| 11 | Bubble-sort+ | Linear-search+ | Bubble-sort | Linear-search |
| 12 | Binary-search+ | Bubble-sort+ | Binary-search | Bubble-sort |

```
#include <stdio.h>

int main(void) {
  int sort[10] = {1, 7, 4, 10, 9, 8, 2, 5, 3, 6};
  int i, j, length = 10, temp;

  for(i = 0; i < length; i++) {
    for(j = length-1; j > i; j--) {
      if (sort[j] < sort[j-1]) {
        temp = sort[j];
        sort[j] = sort[j-1];
        sort[j-1] = temp;
      }
    }
  }

  return 0;
}
```

**Fig. 6** Sample program we prepared for the bubble-sort

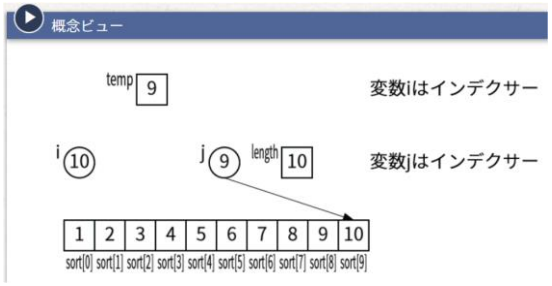


**Fig. 7** Sample PV for the bubble-sort

task" for each task of each participant. In the tasks with our system, the participants generated drawing rules using our system and then manually modified them. They adjusted the drawing positions of arrays and variables being processed by each program, confirmed the specification of the use of each variable as an indexer with our system, and then highlighted expressions such as changing the color of drawing objects by modifying the rule file generated by our system. In the non-system tasks, participants manually defined the drawing rules from scratch. We measured the time taken by each participant to define the appropriate drawing rules. We provided the participants with two tasks at a time and set a maximum of 60 min to complete the two PV creations. Participants who had not completed the two tasks after 60 min were asked to terminate the task at that point. After a 10-min interval, the participants were asked to create another two PVs again for a maximum of 60 min. As shown in Table 1, each subject used our system for either of the 60-min PV creation tasks.

## Evaluation results

The experimental results are presented in Table 2. The numbers in each column indicate the time required to complete the PV-creation task for each target program. "N/A" means the task was not completed within 60 min. The underlined values indicate the time required to complete the task using the proposed system. The results revealed that, for participants who performed the task without the system followed by the task with the system, the times for PV creation were reduced appreciably. We set up 24 pairs of data points, each pair consisting of the time taken to complete two tasks for the same target program, i.e., the time taken to complete a task with the proposed system and the time taken to complete a task without it. Except for two pairs of data points that include the time taken to complete tasks that were not completed in 60 min, the respective times on each pair of data points can be considered to follow a normal distribution. We conducted a paired *t*-test for the 22 paired data points and found a significant difference between the time taken to complete the tasks with and without the proposed system ($t(21) = 4.23$, $p < 0.001$). It suggests that our system has a certain effect on improving the effortlessness of PV creation.

**Table 2** Measured times (s) for PV creation with and without our system

| Participants # | 1st task | 2nd task | 3rd task | 4th task |
|---|---|---|---|---|
| 1 | 1890 | 1018 | 277 | 316 |
| 2 | 3600 | N/A | 474 | 317 |
| 3 | 1194 | 812 | 313 | 441 |
| 4 | 2035 | 922 | 662 | 422 |
| 5 | 3527 | N/A | 1005 | 279 |
| 6 | 1969 | 1061 | 730 | 317 |
| 7 | 1026 | 478 | 1536 | 600 |
| 8 | 1244 | 231 | 1693 | 272 |
| 9 | 1623 | 570 | 1526 | 632 |
| 10 | 1650 | 570 | 1583 | 649 |
| 11 | 1886 | 326 | 2625 | 975 |
| 12 | 1592 | 750 | 3038 | 562 |

However, the decrease in task time for participants who performed the task using our system was generally small. Participants #9 and #10 completed the non-system task for the first program (1st task) in less time than the system task (3rd task). Similarly, participant #12 completed the non-system task for the second program (2nd task) in less time than the system task (4th task). These participants performed the system tasks prior to the non-system tasks; hence, it is considered that experience with the PV definitions could affect these results. That is, the earlier the preceding task, the more unfamiliar the participants tended to use our system, and the later the task, the more familiar they tended to be with defining drawing rules. These results indicate that the proposed system does not have a sufficient effect when the same PV creations are repeated.

## Effortless PV creation based on menu operations for PV definitions

### Approach

Following the results described in the previous section, we extended the dialog interface to further improve the effectiveness of our system for effortless PV creation. The basic approach of the extension is to allow more attributes of drawing objects to be given as parameters by elaborating on the interface that can only input the coordinates of the drawing objects and whether it is an indexer. In accordance with this basic idea, we extended our system to allow the following parameters to be input.

- Conditional expressions in the conditional part
- Label object creation in PV
- Border line colors, background colors, and inner character colors of drawing objects
- Margin sizes in drawing object arrangement
- Layout of array objects (vertical or horizontal)
- Shape of the connector objects representing an indexer (arrow or line)

This approach provides more elaborate support for defining drawing rules in PV creation with TEDViT compared to the system described in the previous section, which suggests drawing objects as typical visualizations for novice learners. By allowing teachers to select expressions that can be described in ruleset files with combo boxes and by limiting the range of possible descriptions to input forms, this approach provides certain parts of rule definitions in TEDViT as interactions using a dialog interface rather than by editing ruleset files. The extended dialog interface can be considered a guide for PV creation. This is expected to reduce workload by mouse operations and cognitive load by the guide and be more effective in improving effortlessness.

Figures 8 and 9 show screenshots of the extended dialog interface. The screen size of the

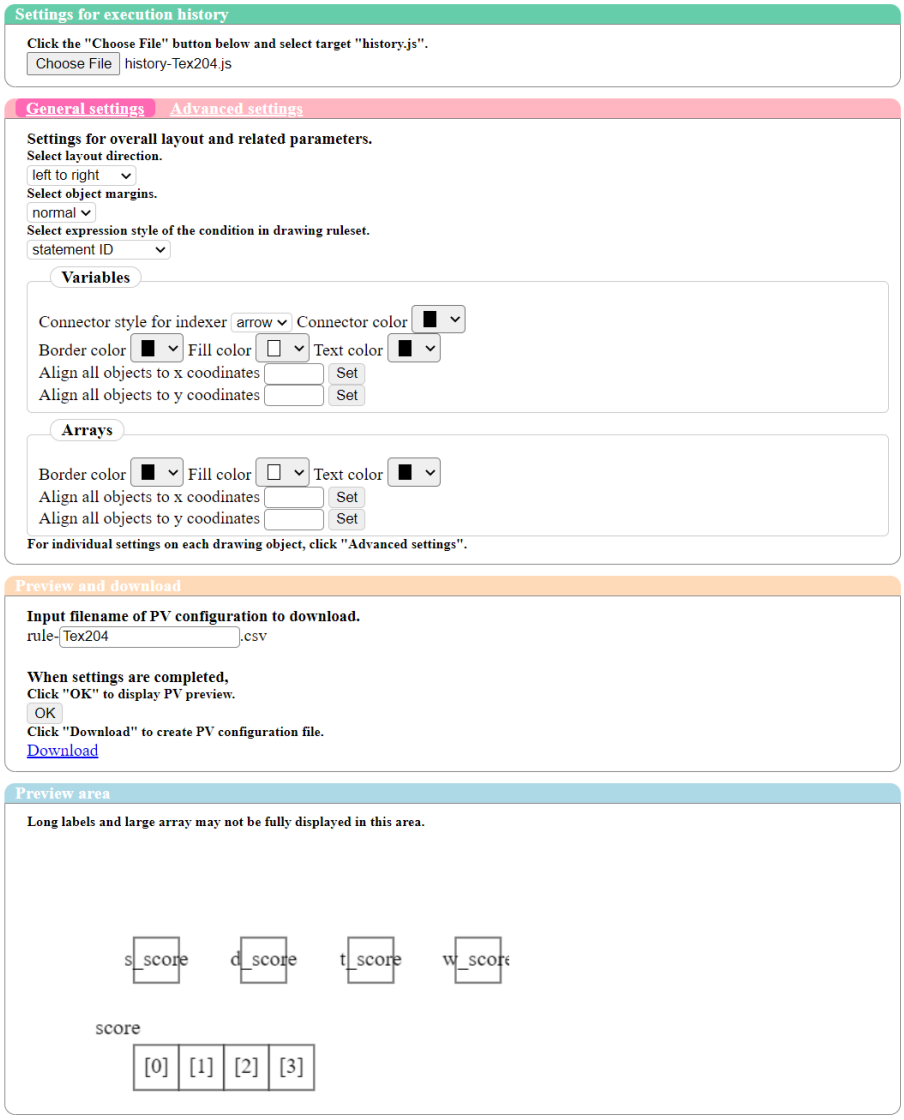


**Fig. 8** Screenshot of the extended dialog interface (general settings)

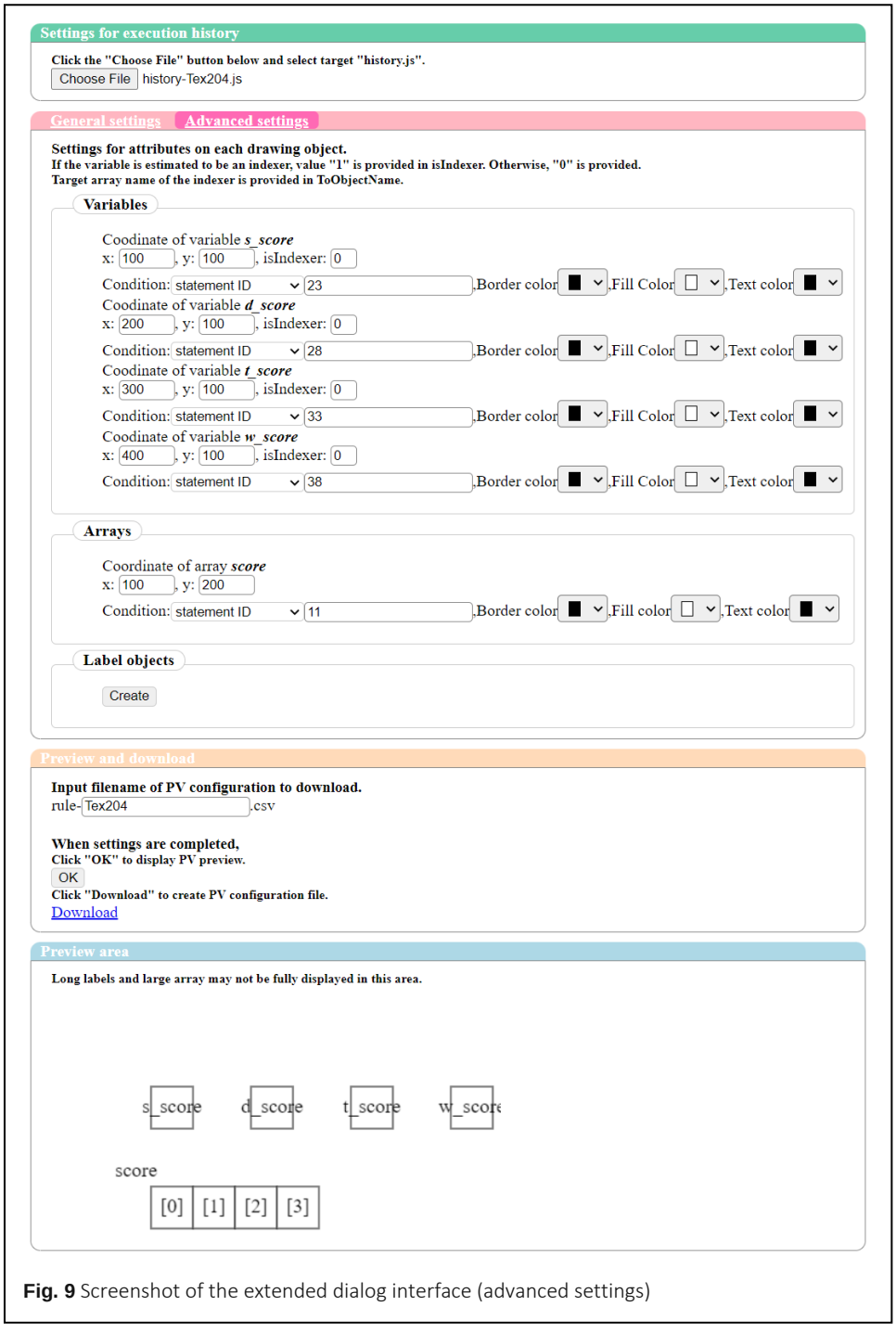


**Fig. 9** Screenshot of the extended dialog interface (advanced settings)

extended interface was larger than that of the interface described in the previous section because more detailed parameters were available. Therefore, we implemented a tab function to switch between general settings for the entire PV (Figure 8) and advanced settings for individual drawing objects (Figure 9). In general settings, a function to set the attribute values of all the drawing objects of variables and arrays simultaneously is included.

Because many parts of the rule definitions can be completed using our extended interface, we implemented a PV preview area in which teachers could check the PV with current parameter settings. A preview of the PV corresponding to the current setting was visualized in the preview area by clicking the "OK" button at the bottom of the interface screen.

## Evaluation experiment

To evaluate the effectiveness of PV creation using our extended interface, we conducted an experiment to measure the actual time required to create PVs using the extended system, as described in the previous section. Nine participants were involved in this experiment: two teachers with over 10 years of experience in teaching programming at university, four 23-years-old CS master's students with experience as programming teaching assistants, and three 22-years-old undergraduate students majoring in CS with the same level of programming experience as the teaching assistants. Although two teachers and one teaching assistant participated in the experiment described in the previous section, we consider that their participation had little influence in both experiments because the interval between the experiments was more than 10 months. We prepared two sample programs and sample PVs as PV creation targets: a binary-search program and a selection-sort program. Each participant underwent two sample programs and PVs and was asked to create the same PVs as the samples. We asked them to create PVs with the extended system and the system described in the previous section for each sample program; hence, the participants performed four tasks in total: binary-search with the extended interface, selection-sort with the extended interface, binary-search with the previous interface, and selection-sort with the previous interface. The extended interface was developed to avoid replacing the system described in the previous section but aimed to improve effortlessness by allowing more PV customizations to be completed on the dialog interface than on the system described in the previous section. For this reason, we set up an experiment in which the time required for PV creation was compared between the extended interface and the system described in the previous section, thereby clarifying the effectiveness of our extension.

We specified the order of the tasks for each participant differently. A list of conditions for each participant is presented in Table 3. In Table 3, items with "+" at the end of the target program indicate tasks using previous interface and items with "++" indicate tasks using the extended interface. Figure 10 shows the sample program prepared for the binary-search, and Figure 11 shows a sample PV for it. We also prepared sample sets of similar complexity for the selection-sort; however, we omit them here.

**Table 3** Conditions per each participant in the experiment with the extended interface ("+" means a task with the dialog interface described in the previous section and "++" means a task with the extended interface)

| Participants # | 1st task | 2nd task | 3rd task | 4th task |
|---|---|---|---|---|
| 1 | Binary-search+ | Selection-sort++ | Selection-sort+ | Binary-search++ |
| 2 | Selection-sort+ | Binary-search+ | Binary-search++ | Selection-sort++ |
| 3 | Selection-sort++ | Binary-search++ | Binary-search+ | Selection-sort+ |
| 4 | Binary-search++ | Selection-sort+ | Selection-sort++ | Binary-search+ |
| 5 | Binary-search+ | Selection-sort++ | Selection-sort+ | Binary-search++ |
| 6 | Selection-sort+ | Binary-search+ | Binary-search++ | Selection-sort++ |
| 7 | Selection-sort++ | Binary-search++ | Binary-search+ | Selection-sort+ |
| 8 | Binary-search++ | Selection-sort+ | Selection-sort++ | Binary-search+ |
| 9 | Binary-search+ | Selection-sort++ | Selection-sort+ | Binary-search++ |

```c
#include <stdio.h>

int main(void) {
  int a[7] = {1, 2, 5, 7, 10, 11, 15};
  int low = 0, mid, high = 6, value = 11;

  while(low <= high) {
    mid = (low + high) / 2;
    if (a[mid] == value) { return 0; }
    else if (a[mid] < value) { low = mid + 1; }
    else { high = mid - 1; }
  }

  return 0;
}
```

**Fig. 10** Sample program we prepared for the binary-search

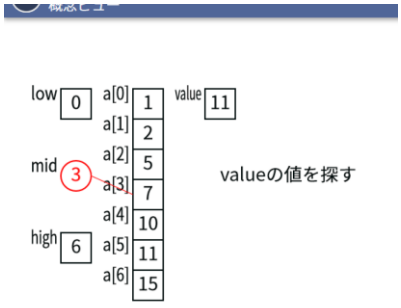


**Fig. 11** Sample PV for the binary-search

We began this experiment by explaining to the participants (for 10 min) the specification of the TEDViT drawing rules and how to use our extended and previous interfaces. Next, we gave them two sample programs, sample PVs, and a dialog interface to use without disclosing the drawing rules and asked them to "define drawing rules to reproduce the same PV as the provided sample PV with the provided interface." The participants generated drawing rules using the provided interface and then manually modified them. In the tasks with the previous interface, the participants adjusted the drawing positions of arrays and variables being processed by each program, confirmed the specification of the use of each variable as an indexer, and then modified the rule files generated by the previous interface to create PV, as in the previous section. In the tasks with the extended interface, the participants adjusted further attributes, including highlighting expressions, such as changing the color of each drawing object on the interface. They modified the rule file generated by the extended interface after specifying as many attributes as possible on the interface. Although each PV used in the experiment could be completely defined only with the specifications on the extended interface, all the participants obtained the rule with some of the specifications missing and hence required some manual modifications of the rule files because of their unfamiliarity with the extended interface. We measured the time taken by each participant to define the appropriate drawing rules. The participants worked on the four tasks in the order listed in Table 3, where the maximum time for each task was 30 min.

## Evaluation results

Table 4 summarizes the measured times. The numbers in each column indicate the number of seconds required to complete each PV-creation task. The underlined values indicate the time required to complete the task using the extended interface.

This result reveals that the time required for PV creation with the extended system tends to be less than that with the previous system. While almost all participants additionally edited the CSV file generated by the previous system in the case of tasks using the previous interface, many of the participants completed PV creation using only the interface in the tasks using the extended interface. The shorter time for PV creation with the extended

**Table 4** Measured times (s) for four PV creation tasks

| Participants # | Binary-search+ | Binary-search++ | Selection-sort+ | Selection-sort++ |
|---|---|---|---|---|
| 1 | 1621 | 189 | 1724 | 572 |
| 2 | 556 | 275 | 1528 | 150 |
| 3 | 733 | 199 | 623 | 397 |
| 4 | 577 | 549 | 1381 | 322 |
| 5 | 607 | 259 | 998 | 263 |
| 6 | 735 | 299 | 1783 | 250 |
| 7 | 432 | 337 | 947 | 486 |
| 8 | 559 | 737 | 1222 | 331 |
| 9 | 923 | 227 | 751 | 332 |

interface is considered to derive from this tendency in task performance; thus, this result suggests that the extended interface further improves the effortlessness of PV creation. We conducted a paired *t*-test for the time spent on the tasks with the previous interface and the time spent on the tasks with the extended interface and found a significant difference between conditions ($t(17) = 5.38$, $p < 0.0001$). Participant #8 spent more time on PV creation for a binary-search using the extended interface. We consider that this was derived from his experience with PV creation: PV creation using the extended interface was his first task, and PV creation using the previous interface was his last task. We conducted a brief interview survey of the participants after the experiment and asked them which function was most useful in the extended interface. Six participants indicated that the PV preview function was the most common. This suggests that visual support is necessary for PV creation, in addition to the support provided by dialog interfaces, where the attributes of the drawing objects are provided as values.

## Effortless PV creation based on the WYSIWYG PV editor

### Approach

To support the interaction between teachers and PV content, some existing systems have functions that design PVs using a GUI. Based on the WYSIWYG AV editor implementation, Karavirta et al. (2002) evaluated the effortlessness of the existing AV systems. Using TEDViT, Tezuka et al. (2016) developed a GUI system that visually defines the positions and attributes of drawn objects to reduce the cost of defining the drawing rules. Hereafter, this study refers to their system as the Tezuka GUI, and Figure 12 is a screenshot of this system.

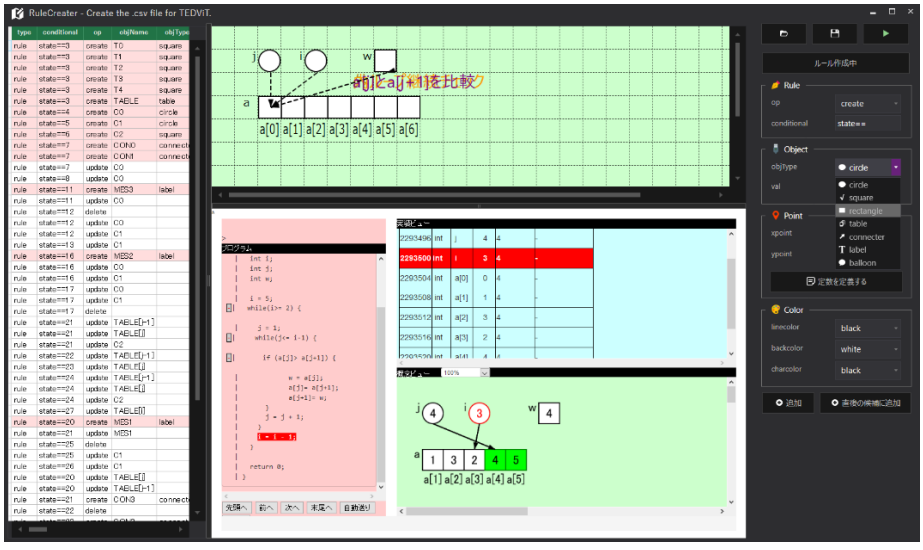


**Fig. 12** Screenshot of the Tezuka GUI

The Tezuka GUI has functions that visually specify the positions of drawn objects, list their available attributes (line color, background color, character color, etc.), specify the values in a combo-box style, and highlight grammatical errors in drawing rules. Tezuka et al. (2016) evaluated the extent to which the Tezuka GUI improved effortlessness (measured using the time needed to create drawing rules for TEDViT) and found that the measured times for rule creation were approximately 40% less with the Tezuka GUI than without.

However, the Tezuka GUI and existing WYSIWYG PV editors support only single PV drawings. In general, PV systems change the drawing content during the program execution process. This allows learners to understand the function of each statement in the target program by observing the differences between PVs. Hence, we regard PVs as visualizations of the target domain world. The meaning of each statement in the program is defined by the extent to which executing the statement changes the target domain world. Importantly, PVs are not simply drawings of data structures but the sequences of drawings linked to program-execution processes. In other words, a single PV is not sufficient for program understanding. A PV would have a certain effect when multiple PVs are sequenced along a time series during the program execution process.

PV dynamics are a direct representation of computer program dynamics that reveal the trajectory of changes in a computer's internal state, such as continuous changes in data structures. By showing these changes directly, dynamic visualizations can offload a learner's cognitive working memory, potentially enabling deeper cognitive processes. Dynamic visualizations can also facilitate cognitive processes that would otherwise require considerable effort (Kühl et al., 2011; Schnotz & Rasch, 2005). In other words, the learning effect of PV systems can be attributed to their dynamic nature. However, PV editors only support the creation of static visualizations. We believe that PV creation cannot be fully supported by single PV drawings. Instead, a function that helps capture the time-series sequence of the PVs is required.

Based on this consideration, we aim to support PV creation more effectively by developing a GUI system that includes PV time-series information. Figure 13 shows a screenshot of the proposed GUI system. Our GUI system consisted of nine display areas, as shown in Figure 14.

PV creators (i.e., teachers) can arbitrarily change the current PV within a time-series PV sequence by dragging the seek bar. PV thumbnails can make creators aware of the PV continuity. PV creators also visually confirmed the differences between adjacent PVs in a time series. The seek bar not only helps PV creators navigate the execution process but also helps them grasp the approximate position of the current PV in a time series. We intend to improve PV creation efficiency using this feature.

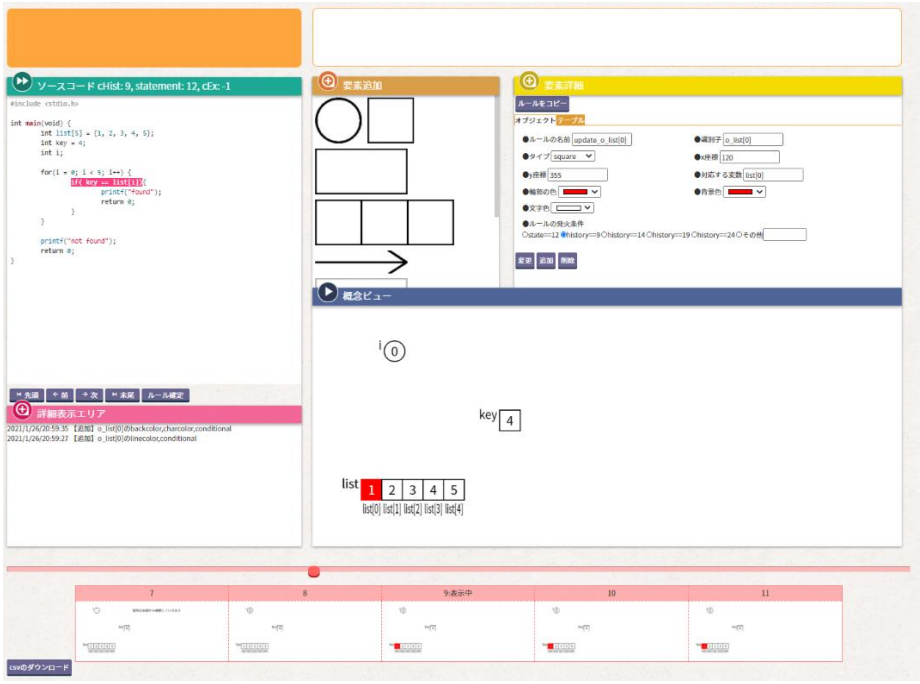


**Fig. 13** Screenshot of our GUI system

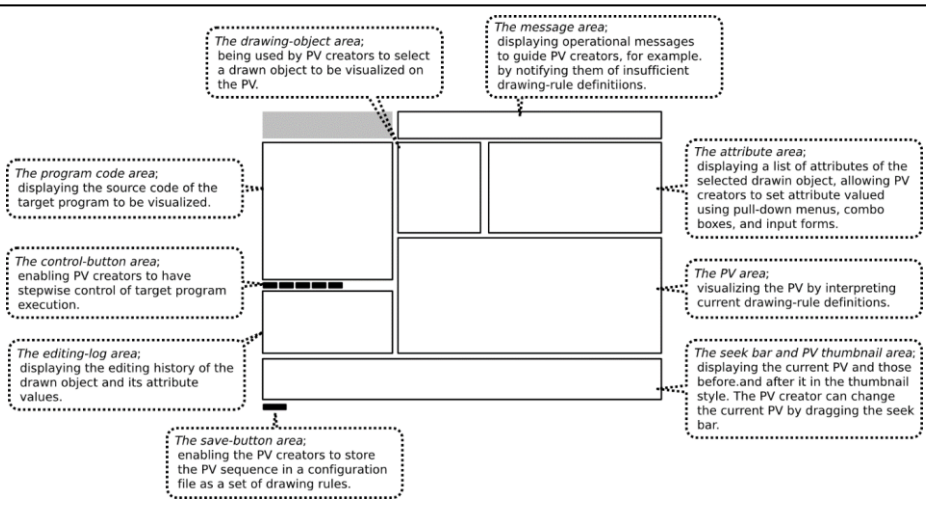


**Fig. 14** Nine display areas in our GUI system

## Evaluation experiment

To evaluate the effectiveness of PV creation using our GUI system, we conducted an experiment, as described in the previous sections, to measure the actual time required to create or modify PVs. The present experiment measured the time required for PV creation using the Tezuka GUI and evaluated the degree of improvement in the effortlessness of our system. Ten participants were involved in this experiment: two teachers with over 10 years of experience in teaching programming at university, five 23-years-old CS master's

students with experience as programming teaching assistants, and three 22-years-old undergraduate students majoring in CS with the same level of programming experience as the teaching assistants. We prepared two sample programs and sample PVs as PV creation targets: a linear-search program and maximum-value derivation program. Each participant received a sample program and was asked to either create the same PV as the sample or modify the provided PV to reproduce the same PV as the sample. To reduce order effects, we first specified whether the participants would use the Tezuka GUI or the proposed system. Table 5 summarizes the conditions of each participant. Figure 15 shows the sample program prepared for a linear-search. We also prepared a sample program of similar complexity for the maximum-value derivation, but we omit it here.

We began this experiment by giving a 45-min explanation to the participants on the specification of the TEDViT drawing rules and how to use the two GUI systems. The reason this approach had a longer explanation time than the evaluation procedures in the previous two sections is that functions provided by GUIs are more complex than the dialog interface, and we had to explain how to use both the Tezuka GUI and our GUI. To explain the two interfaces, we used a binary-search program that differed from the sample programs

**Table 5** Conditions for each participant in the experiment with the GUI interface

| Participants # | Target task | Operation | First use | Second use |
|---|---|---|---|---|
| 1 | Linear-search | Creation | Tezuka GUI | Our GUI |
| 2 | Linear-search | Modification | Tezuka GUI | Our GUI |
| 3 | Finding Maximum | Creation | Tezuka GUI | Our GUI |
| 4 | Finding Maximum | Modification | Tezuka GUI | Our GUI |
| 5 | Linear-search | Creation | Our GUI | Tezuka GUI |
| 6 | Linear-search | Modification | Our GUI | Tezuka GUI |
| 7 | Finding Maximum | Creation | Our GUI | Tezuka GUI |
| 8 | Finding Maximum | Modification | Our GUI | Tezuka GUI |
| 9 | Linear-search | Creation | Tezuka GUI | Our GUI |
| 10 | Finding Maximum | Creation | Our GUI | Tezuka GUI |

```
#include <stdio.h>

int main(void) {
  int list[5] = {1, 2, 3, 4, 5};
  int key = 4, i;

  for(i = 0; i < 5; i++) {
    if (key == list[i]) { printf(“found”); return 0; }
  }
  printf(“not found”);

  return 0;
}
```

**Fig. 15** Sample program we prepared for the linear-search

used in the experiment. Figure 16 shows the program code used in the explanations, and Figure 17 shows a screenshot of the GUI provided to the subjects during the explanations. The same program was used to explain the Tezuka GUI.

```
#include <stdio.h>

int main(void) {
  int search[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11};
  int min = 0;
  int max = 9;
  int middle;

  int a = 1;
  while(min <= max) {
    middle = (min + max) / 2;
    if (search[middle] == a) {
      return 0;
    } else if (search[middle] < a) {
      min = middle + 1;
    } else if (search[middle] > a) {
      min = middle – 1;
    }
  }
  return 0;
}
```

**Fig. 16** Binary-search program used in the explanations of the interfaces

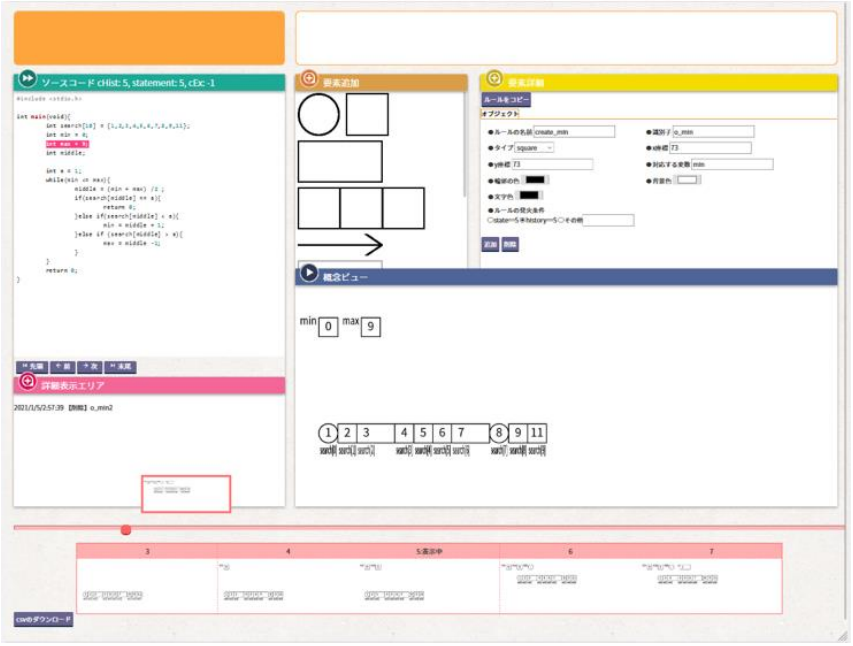


**Fig. 17** Screenshot of our GUI provided in the explanations of interfaces

Next, for the participants assigned to PV creation, we provided a sample program and sample PV that differed from the program used in the explanations of interfaces, without disclosing the drawing rules, and asked them to "define drawing rules to reproduce the same PV as the sample PV." For the participants assigned to the modification, we provided a sample program, sample PV, and a set of drawing rules for the sample PV that included some errors and asked them to "modify the provided drawing rules to reproduce the same PV as sample PV." These procedures were repeated twice for each subject, and the GUI system was changed according to the order listed in Table 5. The participants manipulated the provided GUI system until the PV creation task was complete, specifying the layout of drawing objects, highlighting, and rule-firing conditions to create the rule file. In contrast to the experiments described in Sections "Effortless PV creation based on semi-automatic arrangement of drawing objects" and "Effortless PV creation based on menu operations for PV definitions," in these experiments, none of the participants performed any manual modifications of the rule files obtained from the system.

We measured the time taken by each participant to define appropriate drawing rules for each task. Subsequently, we conducted a questionnaire survey on the effectiveness of the seek bar and PV thumbnail functions using a five-point grading system. We also conducted brief interviews to ascertain participants' opinions regarding the two systems.

## Evaluation results

Table 6 lists the experimental results. Regardless of the order in which they used the GUI systems, target programs, and task operations, all participants took less time to complete the task with our system than with the Tezuka GUI system. The reduction rate based on the average time spent by all participants was 41.3%. We conducted a paired *t*-test for the time spent on the task with Tezuka GUI and the time spent on the task with our GUI, and found a significant difference between conditions ($t(9) = 3.84$, $p < 0.01$). These results suggest that PV creation using our GUI system significantly improves effortlessness. The

**Table 6** Measured Times (s) for PV Creation/Modification with GUI systems

| Participants # | Time with Tezuka GUI | Time with our system |
|---|---|---|
| 1 | 1136 | 925 |
| 2 | 421 | 249 |
| 3 | 1935 | 989 |
| 4 | 1360 | 320 |
| 5 | 914 | 834 |
| 6 | 523 | 416 |
| 7 | 1355 | 845 |
| 8 | 520 | 377 |
| 9 | 2022 | 1015 |
| 10 | 1383 | 819 |

questionnaire survey on the effectiveness of the seek bar and PV thumbnail functions produced an average score of 4.2, suggesting that the participants gave the function a positive rating. In the interview survey, some participants commented that providing PV continuity alongside the time series made it easier to identify errors. Dividing the task times into two groups based on participant operations reduced the average time needed for PV creation and modification by 37.9% and 51.8%, respectively. This suggests that our GUI system supports the task of error correction more effectively.

## Discussion

### Hierarchical integration of our three approaches

As Ihantola et al. (2005) pointed out, effortlessness in the context of AV/PV is a highly subjective matter that includes many factors. Hence, simple comparisons of various approaches are difficult. In this study, we examined three approaches to improve the effortlessness of PV creation: (1) supporting PV creation by semi-automatically arranging drawing objects oriented toward novice learners, (2) supporting it with menu operations with a dialog interface, and (3) supporting it visually with a WYSIWYG PV editor. Although we evaluated these approaches based on the time required for PV creation, it is difficult to discuss the superiority or inferiority of each approach based on the time required for PV creation because the participants and target programs were different.

However, because these three approaches are not mutually exclusive, it is expected that their integration could provide more effective support for PV creation. In particular, the findings from the experimental results described in the previous section may provide some insight into integrating the three approaches; for example, a dialog interface could be used to create general PVs, and a graphical interface could be used to modify and compensate for their shortcomings. Such an integrated approach to support PV creation can be modeled, as shown in Figure 18. The three approaches described in this study do not support PV creation independently and exclusively but can provide hierarchical support by being integrated.

This style of support for PV creation is constructed not only by incorporating the approaches described in this study as layers, but various approaches can also serve as layers constituting hierarchical support. For example, teachers could create PVs from PV

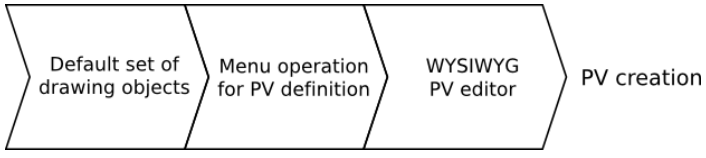


**Fig. 18** Layer structure of support for PV creation

templates for various algorithms described in (Rößling & Ackermann, 2007) and then use the WYSIWYG PV editor to visually modify those PVs. Therefore, to achieve effortless PV creation, it is necessary to consider not only various support approaches independently but also the hierarchical integration of support.

We are currently developing a system that allows CSV files generated by the extended dialog interface described in Section "Effortless PV creation based on menu operations for PV definitions" to be directly input to the graphical interface described in Section "Effortless PV creation based on the WYSIWYG PV editor." We plan to develop PV template collections by investigating various teaching materials for novice programmers. With these plans, we aim to develop a framework to hierarchically improve effortlessness in PV creation by providing various options for support layers. We plan to introduce this framework into actual educational setups to evaluate the long-term effortlessness of PV creation.

## Limitations of our study

In this study, we developed three approaches to reduce the cost of PV creation and evaluated the degree of cost reduction by measuring the time required for PV creation based on each approach. One of the limitations of this study is that the time reduction rates derived from each evaluation experiment were not sufficiently reliable owing to the low sample size. However, we believe that the sample size problem will decrease as we continue our studies, including conducting the same evaluation experiments as in this study. Moreover, as discussed below, it is not clear how general our three approaches are; hence, we consider that it is not important to obtain the exact time reduction rates. We plan to study other approaches for reducing the cost of PV creation by integrating various approaches in layer structures in the future, rather than finding exact time reduction rates, and examining the superiority or inferiority of each approach.

This study did not discuss the general attributes of teachers who benefit from the three approaches in PV creation tasks. The evaluation experiments in Sections "Effortless PV creation based on semi-automatic arrangement of drawing objects," "Effortless PV creation based on menu operations for PV definitions," and "Effortless PV creation based on the WYSIWYG PV editor" were conducted with the participation of teachers with sufficient experience in teaching programming to novice students but were not designed to clarify the extent to which differences in teaching experiences correlate with differences in the cost reduction rates of PV creations. In general, teachers with more teaching experience tend to have more explicit intentions of instruction than those with less experience; hence, we expect a positive correlation between teaching experience and cost-reduction effects. Although examining this hypothesis is difficult because it would require cooperation from

many teachers, we hope to accumulate empirical knowledge about our hypothesis by continuing our efforts.

Furthermore, we must pay attention to the fact that all the approaches described in this study are strongly dependent on the customizability of TEDViT, which is one of the limitations of our study. In other words, all three approaches are based on the use of TEDViT, and the effects of using other PV systems are not clear. Indeed, a few PV systems are known to have the same level of customizability as TEDViT, and it is unclear how much generality each of three approaches have. However, the supporting targets of our three approaches are the PV customizations that are considered to lead to the positive learning effects of TEDViT, and we believe that a certain degree of generality can be recognized in those targets. Our approaches mainly support customizations of the layout of drawing objects, shapes, and colors of each object, and time-series management of PVs. These are the targets for which teachers' customization would improve learning effectiveness in general PVs generated by other existing systems. Although it is difficult to apply the implementations in our three approaches as is, we believe that adopting approaches similar to the ones described in this study in existing PV systems will contribute to improving the program understanding of novice learners and reducing teachers' efforts required to create and customize PVs for novices' understanding.

## Conclusion

In this study, we describe three approaches for improving the effortlessness of PV creation in TEDViT, a PV system we have developed that allows teachers to define PVs based on their intents of instruction and have introduced it in several practical classes. We have observed that program learning using TEDViT has a high learning effect on learners in practical classes (Ihara et al., 2017; Kogure et al., 2018; Yamamoto et al., 2017; Yamashita et al., 2016; Yamashita et al., 2017; Yamashita et al., 2020). Based on these experiences, we believe that the customizability of PVs, including limited approaches such as Jsvee & Kelmu, is a necessary requirement for cultivating learners' understanding of a program. However, PV customization is a time-consuming task for teachers. The high time cost of preparation is considered one of the main obstacles to the continuous use of PV systems in actual classrooms; hence, effortless PV creation is required.

To address this issue, we developed a system that supported PV creation by semi-automatically arranging typical drawing objects as PVs for novice learners. The system has a dialog interface that specified parameters such as the coordinates of the drawing objects and generated PV definition files to allow further PV customizations. Therefore, it is expected to improve the efficiency of PV creation without any loss of customizability. An evaluation experiment was conducted to measure the time required to create PVs using the

proposed system. The experimental results revealed that our system reduced the average task time by at least 48.1%.

Next, we extended the dialog interface and developed an extended system that provides certain PV definitions in TEDViT as menu operations. We expected that our extended interface would reduce the workload of teachers by allowing them to select expressions that can be described in a ruleset file with combo boxes and by limiting the range of possible descriptions to input forms. The evaluation experiment to measure the PV creation time revealed that, compared with the previous system, our extended system reduced the average task time by 65.1%. In addition, the interview survey suggested a need for visual support for PV creation.

Finally, we developed a system that supports PV creation using a WYSIWYG PV editor. Many existing systems, including the Tezuka GUI, support only the drawing of PVs, even though PVs are not simply drawings of data structures but sequences of drawings alongside a program-execution process. Therefore, we developed a PV creation support system that incorporates time-series information into a GUI that considers the continuity of the drawings. We conducted an evaluation experiment to measure the time required to create PVs by using the Tezuka GUI and our GUI. The results revealed that our GUI system reduced the average time by 41.3% compared with the Tezuka GUI.

The evaluation results suggested that each of the three approaches in this study improved the effortlessness of PV creation to a certain degree. Although the time reduction rates obtained from the three evaluation experiments have a limitation in that the sample size is insufficient to provide reliable evaluation results, they are useful because they provide insight into the effect size of each approach. However, we considered that the individual effect size of each approach was not that important. Because there were various perspectives on effortlessness in PV systems, and it was difficult to define effortless PV creation, the effect size expressed in terms of time reduction rate did not have much generality.

The main finding of this study was that several possible approaches that are not mutually exclusive are available, suggesting a certain degree of effectiveness in improving effortlessness. Non-exclusive support for PV creation can be combined in a layered structure, which can further improve effortlessness. It is also possible to add more support layers using various approaches such as PV templates. The integrability defined by Ihantola et al. (2005) was mainly considered between the PV system and educational environment. We conclude that it is necessary to include the integrability between several systems, as has been partially observed in Jsvee & Kelmu.

Future studies will include the investigation of new support layers for PV creation and long-term large-scale evaluation experiments of the integrated support environment. In particular, PV template collections based on various teaching materials for novice

programming learners are expected to be a promising support layer that will improve the effortlessness of PV creation. By continuing these efforts, we aim to develop an environment in which a PV system can be continuously introduced into actual classrooms.

**Abbreviations**

AV: Algorithm visualization; CS: Computer science; CSV: Comma-separated value; GUI: Graphical user interface; PS interaction: Producer-System interaction; PV: Program visualization; TEDViT: Teacher's explaining design visualization tool; VC interaction: Visualization-Consumer interaction.

**Authors' contributions**

KY participated in the development of the systems, summarized the research, and wrote this paper. YS, SK, and YN implemented and evaluated the system based on semi-automatic arrangement of drawing objects. YK, SK, and YN implemented and evaluated the extended system based on menu operations for PV definitions. MS, SK, and YN implemented and evaluated the extended system based on WYSIWYG PV editor. YI, TK, and RY gave advice based on the actual teaching experience. All authors read and approved the final manuscript.

**Funding**

This study was supported by JSPS KAKENHI Grant Numbers JP18K11566, JP18K11567, and JP19K12259.

**Availability of data and materials**

Not applicable.

## Declarations

**Competing interests**

The authors declare that they have no competing interests.

**Author details**

[1] Faculty of Business Administration, Tokoha University, 1230 Miyakoda, Kita-ku, Hamamatsu, Shizuoka 431-2102, Japan.

[2] Faculty of Informatics, Shizuoka University, 3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka 432-8011, Japan.

[3] Faculty of Engineering, Sanyo-Onoda City University, 1-1-1 Daigakudori, Sanyo-Onoda, Yamaguchi 756-0884, Japan.

[4] Shizuoka University, 836 Oya, Suruga-ku, Shizuoka City, Shizuoka 422-8017, Japan.

Received: 17 January 2022 Accepted: 9 December 2022

Published: 28 February 2023 (Online First: 2 January 2023)

**References**

Atemezing, G. A., & Troncy, R. (2014). Towards a linked-data based visualization wizard. In O. Hartig, A. Hogan & J. Sequeda (Eds.), *Proceedings of the 5th International Workshop on Consuming Linked Data* (pp. 1–12). CEUR-WS.org.

Guo, P. J. (2013). Online Python Tutor: Embeddable web-based program visualization for CS education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)* (pp. 579–584). Association for Computing Machinery. https://doi.org/10.1145/2445196.2445368

Helminen, J., & Malmi, L. (2010). Jype – A program visualization and programming exercise tool for Python. In *Proceedings of the 5th International Symposium on Software Visualization* (pp. 153–162). Association for Computing Machinery. https://doi.org/10.1145/1879211.1879234

Ihantola, P., Karavirta, V., Korhonen, A., & Nikander, J. (2005). Taxonomy of effortless creation of algorithm visualizations. In *Proceedings of the First International Workshop on Computing Education Research* (pp. 123–133). Association for Computing Machinery. https://doi.org/10.1145/1089786.1089798

Ihara, D., Kogure, S., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2017). Algorithm learning by comparing visualized behavior of programs. In W. Chen et al. (Eds.), *Proceedings of the 25th International Conference on Computers in Education* (pp. 385–390). Asia-Pacific Society for Computers in Education.

Karavirta, V., Korhonen, A., Nikander, J., & Tenhunen, P. (2002). Effortless creation of algorithm visualization. In *Proceedings of the Second Annual Finnish/Baltic Sea Conference on Computer Science Education* (pp. 52–56). University of Joensuu.

Kogure, S., Fujioka, R., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2014). Code reading environment according to visualizing both variable's memory image and target world's status. In C.-C. Liu et al. (Eds.), *Proceedings of the*

*22nd International Conference on Computers in Education* (pp. 343–348). Asia-Pacific Society for Computers in Education.

Kogure, S., Ogasawara, K., Yamashita, K., Noguchi, Y., Konishi, T., & Itoh, Y. (2019). Application of program learning support system to object-oriented language. In M. Chang, H.-J. So, L.-H. Wong, F.-Y. Yu & J.-L. Shih (Eds.), *Proceedings of the 27th International Conference on Computers in Education* (pp. 348–350). Asia-Pacific Society for Computers in Education.

Kogure, S., Ye, Y., Yamashita, K., Noguchi, Y., Konishi, T., & Itoh, Y. (2018). A learning support system for understanding pointers in C language based on program behavior visualization. In J. C. Yang, M. Chang, L.-H. Wong & M. M. T. Rodrigo (Eds.), *Proceedings of the 26th International Conference on Computers in Education* (pp. 355–357). Asia-Pacific Society for Computers in Education.

Kühl, T., Scheiter, K., Gerjets, P., & Edelmann, J. (2011). The influence of text modality on learning with static and dynamic visualizations. *Computers in Human Behavior*, *27*(1), 29–35. https://doi.org/10.1016/j.chb.2010.05.008

Malone, B., Atkison, T., Kosa, M., & Hadlock, F. (2009). Pedagogically effective effortless algorithm visualization with a PCIL. In *Proceedings of the 39th IEEE International Conference on Frontiers in Education* (pp. 1–6). The Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/FIE.2009.5350481

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot3. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (pp. 373–376). Association for Computing Machinery. https://doi.org/10.1145/989863.989928

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. (2002). Exploring the role of visualization and engagement in computer science education. In *Working Group Reports from the 2002 Conference on Innovation and Technology in Computer Science Education* (pp. 131–152). Association for Computing Machinery. https://doi.org/10.1145/960568.782998

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, *39*(4), 204–223. https://doi.org/10.1145/1345375.1345441

Rößling, G., & Ackermann, T. (2007). A Framework for generating AV content on-the-fly. *Electronic Notes in Theoretical Computer Science*, *178*, 23–31. https://doi.org/10.1016/j.entcs.2007.01.036

Rössling, G., & Freisleben, B. (2002). ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing*, *13*(3), 341–354. https://doi.org/10.1006/jvlc.2002.0239

Roth, S., Hauder, M., Zec, M., Utz, A., & Matthes, F. (2013). Empowering business users to analyze enterprise architectures: Structural model matching to configure visualizations. In *Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference Workshops* (pp. 352–360). The Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/EDOCW.2013.46

Schnotz, W., & Rasch, T. (2005). Enabling, facilitating, and inhibiting effects of animations in multimedia learning: Why reduction of cognitive load can have negative results on learning. *Educational Technology Research and Development*, *53*(3), 47–58. https://doi.org/10.1007/BF02504797

Sirkiä, T. (2018). Jsvee & Kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process*, *30*(2), e1924. https://doi.org/10.1002/smr.1924

Sirkiä, T., & Sorva, J. (2015). Tailoring animations of example programs. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (pp. 147–151). Association for Computing Machinery. https://doi.org/10.1145/2828959.2828965

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, *13*(4), 1–64. https://doi.org/10.1145/2490822

Tezuka, D., Kogure, S., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2016). GUI based environment to support writing and debugging rules for a program visualization tool. Program Visualization Tool. In W. Chen, J.-C. Yang, S. Murthy, S. L. Wong & S. Iyer (Eds.), *Proceedings of the 24th International Conference on Computers in Education* (pp. 303–305). Asia-Pacific Society for Computers in Education.

Tudoreanu, M. E. (2003). Designing effective program visualization tools for reducing user's cognitive effort. In *Proceedings of the 2003 ACM Symposium on Software Visualization* (pp. 105–114). Association for Computing Machinery. https://doi.org/10.1145/774833.774848

Velázquez-Iturbide, J. Á., Pareja-Flores, C., & Urquiza-Fuentes, J. (2008). An approach to effortless construction of program animations. *Computers and Education*, *50*(1), 179–192. https://doi.org/10.1016/j.compedu.2006.04.005

Yamamoto, R., Anzai, Y., Kogure, S., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2017). Learning environment for recursive functions by visualization of execution process. In W. Chen et al. (Eds.), *Proceedings of the 25th International Conference on Computers in Education* (pp. 421–426). Asia-Pacific Society for Computers in Education.

Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2016). Practices of algorithm education based on discovery learning using a program visualization system. *Research and Practice in Technology Enhanced Learning*, *11*(1), 15. https://doi.org/10.1186/s41039-016-0041-5

Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2017). Classroom practice for understanding pointers using learning support system for visualizing memory image and target domain world. *Research and Practice in Technology Enhanced Learning*, *12*(1), 17. https://doi.org/10.1186/s41039-017-0058-4

Yamashita, K., Hiramatsu, Y., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2019). Extending program visualization system based on teacher's intent of instruction to support learning dynamic data structures. In M. Chang, H.-J. So, L.-H. Wong, F.-Y. Yu & J.-L. Shih (Eds.), *Proceedings of the 27th International Conference on Computers in Education* (pp. 354–356). Asia-Pacific Society for Computers in Education.

Yamashita, K., Sakata, K., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2020). Learning support system for understanding pointers based on pair of program visualizations and classroom practices. In H.-J. So et al. (Eds.), *Proceedings of the 28th International Conference on Computers in Education* (pp. 658–663). Asia-Pacific Society for Computers in Education.

Yan, Y., Hiroto, N., Kohei, H., Shota, S., & He, A. (2014). A C programming learning support system and its subjective assessment. In *Proceedings of the 2014 IEEE International Conference on Computer and Information Technology* (pp. 561–566). The Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/CIT.2014.23

## Publisher's Note