PVC: Visualizing C Programs on Web Browsers for Novices*

Ryosuke Ishizue

Department of Computer Science and Engineering
Waseda University
Tokyo, Japan
ishizue@ruri.waseda.jp

Hironori Washizaki

Department of Computer Science and Engineering
Waseda University
Tokyo, Japan
washizaki@waseda.jp

Kazunori Sakamoto

National Institute of Informatics
Tokyo, Japan
exkazuu@nii.ac.jp

Yoshiaki Fukazawa

Department of Computer Science and Engineering
Waseda University
Tokyo, Japan
fukazawa@waseda.jp

ABSTRACT

Many researchers have proposed program visualization tools for memory management because this is a challenging concept for novice programmers. For example, SeeC and PythonTutor (PT) are state-of-the-art tools for C languages. However, three problems hinder the use of these and other tools: capability (P1), installability (P2), and usability (P3). (P1) Tools do not fully support dynamic memory allocation or File Input / Output (I/O) and Standard Input. (P2) Novice programmers often have difficulty installing SeeC due to its dependence on Clang and setting up an offline environment that uses PT. (P3) Revisualization of the modified source code in SeeC requires several steps. To alleviate these issues, we propose a new visualization tool called PlayVisualizerC (PVC). PVC, which is designed for novice C language programmers to provide solutions (S1–3) for P1–3. S1 offers complete support for dynamic memory allocation, standard I/O, and file I/O. S2 involves installation in a user web browser and its server program is initiated by executing a jar file. S3 reduces the steps required for revisualization. To evaluate PVC, we conducted an experiment and questionnaire involving 30 students. Students using PVC solved a set of four programming tasks on average 1.7 times faster and with 19% more correct answers than those using a current state-of-the-art visualization tool.

CCS CONCEPTS

- **Human-centered computing** → *Visualization systems and tools*;
- **Social and professional topics** → *Computer science education*;

KEYWORDS

C language, visualization, web application

*This paper is an extended version of a poster "An Interactive Web Application Visualizing Memory Space For Novice C Programmers"[11] presented at the 48th ACM Technical Symposium on Computer Science Education (SIGCSE 2017).

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGCSE '18, Feb. 21–24, 2018, Baltimore, MD, USA

© 2018 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5103-4/18/02.

<https://doi.org/10.1145/3159450.3159566>

ACM Reference Format:

Ryosuke Ishizue, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. 2018. PVC: Visualizing C Programs on Web Browsers for Novices. In *SIGCSE '18: The 49th ACM Technical Symposium on Computer Science Education, Feb. 21–24, 2018, Baltimore, MD, USA*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3159450.3159566>

1 INTRODUCTION

Various visualization techniques have been proposed to aid programmers in understanding the program execution status [1, 3, 7, 18, 21]. Most existing debuggers and integrated development environments such as GDB and Eclipse provide limited features to visualize the program execution status. Typically, these applications display simple text outputs, but do not visualize the relationships between variables, pointers, and memory. Learning how to use these tools is often difficult for novice programmers (hereafter referred to as novices)¹. Instead of enhancing the understanding of programming languages, these tools often hinder novices.

C programming language (C language) is popular and is typically one of the first languages learned by novices. However, mastery of C language requires that users learn a basic but difficult-to-grasp concept of memory management, including pointers and dynamic memory allocation. This can be extremely challenging for novices [2, 14, 17].

To assist novices, previous studies have proposed tools to visualize the program execution status [6, 12, 13, 20]. For example, SeeC and Python Tutor (PT) are state-of-the-art tools that effectively visualize C programs. However, three problems affect these and other tools: **capability (P1)**, **installability (P2)**, and **usability (P3)**. (P1) SeeC does not fully support dynamic memory allocation. It displays the size of the allocated memory in bytes, but omits more detailed memory values. PT does not support file Input / Output (I/O) and standard Input. (P2) It is difficult for novices to install SeeC due to its dependency on Clang, a compiler. Moreover, students struggle to use PT in an offline environment, and teachers must setup PT for C language on their computer. (P3) SeeC is a desktop application consisting of a compiler and a visualizer. Users cannot modify the source code during SeeC's visualization. Thus, SeeC requires many steps to modify and revisualize C programs. PT

¹A **novice** is defined as a person learning C language who is currently trying to understand the concept of memory management, including pointers and dynamic memory allocation.

Figure 1: Screenshots of SeeC (left) and PT (right).

Figure 2: Overview of the PVC architecture

does not have this problem because it is a web application running on a browser window.

To solve P1–3, we propose PlayVisualizerC (PVC), which is an interpreter of C language with features to visualize the program execution status. PVC provides three solutions (S1–3) to the problems outlined above. (S1) PVC can fully visualize variables, pointers, arrays, dynamically allocated memory, and their relationships. It supports standard I/O and file I/O. (S2) PVC is implemented as a web application. Because installation requires only a browser, executing a jar file can start the server program. (S3) PVC allows users to revisualize a program after source code modifications using a single button click.

The novel contributions of this paper are as follows:

- (1) PVC can aid novices in understanding the program execution status and behaviors.
- (2) PVC is published as a web application at <http://play-visualizer-c.herokuapp.com>, and as a standalone version at <https://github.com/Ryoskate/PlayVisualizerC/releases>.
- (3) We conducted an experiment and a questionnaire to verify that PVC addresses P1–3.

Figure 3: Screenshot of the PVC interface

2 RELATED WORKS

Previous studies have indicated that the concepts of memory and pointer are extremely challenging for novices. For example, Lahtinen et al. investigated the difficulties faced by novices in learning programming languages [14] and found that more than 500 students indicated that finding bugs in a program is the most difficult part of learning to program. Moreover, pointers and references are identified as two of the hardest programming concepts.

Milne and Petersen also investigated challenges surrounding learning and teaching of object-oriented programming (OOP) [17]. Their questionnaire showed that learning copy constructor and virtual function is difficult, primarily due to the fact that memory and pointer are hard concepts to grasp. These studies reveal that

Table 1: Comparison of PVC, SeeC, and PT functionality

Comparison items	PVC	SeeC	PT
Summary			
Capability (P1)	OK	(NG) It does not support malloc.	(NG) It does not support I/O.
Installability for client (P2)	(OK) Only a browser	(NG) Difficult to setup	(OK) Only a browser
Installability for server (P2)	(OK) Launch the playvisualizer.jar file.	(NG) Not supported (Desktop application)	(NG) Very difficult to setup
Usability (P3)	(OK) Open in a browser, write the code, and press execute button.	(NG) Compiling is necessary beforehand. Visualizer cannot edit the code.	(OK) Open in a browser, write the code, and press execute button.
Language feature support			
malloc	Y	N	Y
I/O	Y	Y	N
Visualization expression			
Variables grouped by stack	Y	Y	
Pointer references represented by arrows	Y	Y	Y
Addresses of variables shown	Y	N	N
Display values of uninitialized variables	Random number	! or ? mark	An emoji
Visualization method	Run and create trace data using a proprietary execution environment (Junicoen).	Create execution trace data using LLVM (Clang).	Create execution trace data using GCC and a hacked Valgrind.

understanding memory and pointer are important steps towards progressing in programming ability.

Many studies have proposed visualization techniques and tools, which can effectively aid novices' understanding of these programming concepts. Some research has focused on visualizing the memory state of C language. Koike and Go proposed SuZMe, which visualizes the memory state by byte unit with a horizontal straight line [13]. SuZMe also checks the values of variables and memory allocation in detail. Milne et al. proposed a program visualization tool named OGRE [18]. OGRE generates planes to represent memory space such as Global, Heap, main(), and function(). An object is represented as a figure within three-dimensional space and cylinders connecting figures represent references. Egan and McDonald proposed SeeC, which visualizes the running state of a C program [6]. Moreno et al. proposed Eliot, which is an interactive animation environment to visualize algorithms written in the C programming language [15].

Other tools such as Jeliot [19] and OOP-anime [7] support OOP languages such as Java. These tools show the relationship between class, instance, and variable with figures such as lines and boxes.

Similar to our efforts for PVC, previous studies have developed these tools as web applications to improve accessibility. One example is Python Tutor (PT), which is a program visualization tool proposed by Guo that specializes in supporting Python and the embeddability in web-pages [9, 10]. Currently, PT supports visualization of other languages, including C language. Our literature investigation revealed that SeeC and PT are currently the most useful and widely available state-of-the-art tools for novices to learn about memory management in C language.

3 PROBLEMS WITH STATE-OF-THE-ART TOOLS

Figure 1 (left) and 1 (right) show the visualization results of SeeC and PT, respectively. The effectiveness of these two visualization expressions are similar. They have three common problems: **capability (P1)**, **installability (P2)**, and **usability (P3)**. Table 1 summarizes these programs and compares the features of PVC, SeeC, and PT.

Capability (P1): SeeC does not support dynamic memory allocation (e.g., malloc) completely. Instead it shows the byte size of dynamically allocated memory in text. Figure 1 (left) illustrates an example of a visualization result from SeeC, which includes only the line *< n bytes of dynamically allocated memory >*. As this example shows, SeeC users cannot see the content of the memory allocated by programs such as *malloc*. On the other hand, PT does not allow file I/O and standard Input due to security issues. It is inconvenient that these functions cannot be used.

Installability (P2): The dependency of SeeC on Clang makes installation somewhat difficult. Many novice-oriented visualization tools proposed in previous studies require users to setup an execution environment. This is the main obstacle preventing users from installing these kinds of tools, regardless of a tool's usefulness [5]. Moreover, these tools are usually restricted to specific operating systems.

Another problem with PT is installability on a server computer. Even if students or teachers try to use PT, setting up a server program for C language on their own computer can be burdensome.² The other problem is the stability of the connection with the server. Users may not be able to visualize with PT due to the internet connection environment. For example, one study found that PT could not send back the visualization result when the client was connected to the Internet via *eduroam*, a famous academic Internet service.³

For these reasons, SeeC and PT are insufficient for novices to write and visualize their programs in class.

Usability (P3): Users cannot modify source code during SeeC's visualization because the source code is shown in a read-only text area. To revisualize a program, SeeC requires users to execute the following four steps each time the source code is modified:

²Github of PT says "it can be hard to run your own visualizer locally for non-Python languages, since there are complex setups in v4-cokapi/ that I haven't yet cleanly packaged up."

³Eduroam is an academic wireless LAN roaming infrastructure developed in GEANT (formerly TERENA) in Europe. It is the de facto standard of campus wireless LAN in 80 countries and regions in the world [8]. It has some limitations such as port restrictions.

- (1) Modify the source code file with an external editing application.
- (2) Recompile to generate an executable file with SeeC on a terminal or command prompt.
- (3) Open the executable file to generate a recording file for program behavior and execution status visualization.
- (4) Open the recording file in SeeC's viewer.

PT does not have such a problem.

4 OVERVIEW OF PVC

We implemented PVC with HTML5, JavaScript, C parser, and a semantic analyzer to overcome **P1–3**. The C parser and the semantic analyzer work on the server side. Figure 2 systematically overviews the program, while Fig. 3 shows a screenshot of PVC. To implement our design, we used *ANTLR*, which parses the source code, and *Junicoen*, which creates and executes the abstract syntax tree (AST), *UniTree*, in C language on the server [16]. PVC is a web application that simply requires users to input the URL into a web browser. Additionally, we provide a standalone version that launches a server-side system running on Java on a local computer. The standalone version does not require an internet connection. Therefore, PVC provides a solution (**S2**) to **installability (P2)**.

4.1 Tool Usage

Figure 3 shows an execution example in PVC. PVC has five GUI components: (1) editor, (2) execution controller buttons, (3) I/O window, (4) canvas for visualization, and (5) file upload form. Users can write source code in the editor. Clicking on the execution control buttons initiates the step execution. The I/O window shows the content of the standard output written by the program (e.g., *printf*) and accepts standard input (e.g., *scanf*). Canvas shows the program's execution status using tables and figures. PVC adaptively changes its layout to correspond with the size of the browser window.

The top of the GUI in Fig. 3 shows the program's execution controller. The controller includes the following six buttons: (i) change editor font size, (ii) initiate program execution, (iii) stop program execution, (iv) go backward for all step, (v) go backward one step, (vi) go forward one step, and (vii) go forward all steps. A statement unit executes each step.

Users also can upload files to a temporary space on the server side created for each browser session from the user files form. They can use these files via programs such as *fgets*, *fputc*.

PVC requires three steps:

- (1) Input the URL of PVC into a web browser.
- (2) Insert the source code into the editor to visualize it.
- (3) Press the button to execute the program.

Simply changing the code and pressing the execution button during visualization allows the users to change the program. Thus, PVC provides a solution (**S3**) to **usability (P3)**.

4.2 Visualization Features

PVC, SeeC, and PT use a similar approach for visualization. The programs show values, names, and types of variables using arrows and boxes to represent pointer and stack references. For example, colored arrows and variable addresses help users understand that

Table 2: Examination questions (T1–4) and questions in the questionnaire about usefulness (Q1–4)

T1	What are the final values of variables a, b, c, d, e?
T2	What are the values of variables n, r, (*pn) when the function f is returned for a third time?
T3	What are the final values of heap memory, which has not been freed, and of the pointer variable, which refers to the memory when the main function return?
T4	When are the values of n=1, a='B', b='A', c='C'?
Q1	Do you think this visualization tool helps solve the problems in the experiment?
Q2	Do you think this visualization tool is useful for C language novices?
Q3	Do you think this visualization tool is more accessible than other visualization or debugging tools (e.g., Eclipse, Visual Studio, GDB) that you have used?
Q4	In which areas of C language is this visualization program a useful learning tool? (multiple answers allowed) (A. variable, B. pointer, C. array, D. function, E. dynamic memory allocation, F. structure, G. data structures and algorithms, H. other)

references differ. Movable figures make it easier to see the visualized results, while the display depth of the recursive functions informs users of how many times it has been called.

Only PVC supports the following important features:

- Dynamically allocated memory visualized⁴⁾
- Memory addresses value displayed in hexadecimal
- File and standard I/O

Figure 3 presents an example of visualization with PVC. There are two boxes on the canvas: *main* and *GLOBAL*. *main* represents a stack of the main function, while *GLOBAL* contains dynamic variables for heap memory allocated by *malloc*. The table columns in the box show (1) type, (2) name, (3) value, and (4) address of each variable. For example, the first line of *main* refers to a variable of *n*, which is a type *int* and has a value of 3. Moreover, some variables, such as *ptr* and *d_arr*, refer to other variables. The pointer reference is represented as an arrow with the same color as the reference address (e.g., the green arrow from the row containing *ptr* to the row containing *arr*[2]).

Figures 1 (left) and 3 visualize the same code. Although SeeC stops its visualization, PVC visualizes the allocated heap memory referred to by the pointer variable *d_array*, which is not free. Therefore, PVC can be useful for debugging a program with memory leak action calls. Consequently, PVC provides a solution (**S1**) to **capability (P1)**. Thus, PVC can visualize pointer and dynamic memory allocation (Fig. 3). These solutions are summarized in Table 1.

5 EXPERIMENT

We investigated the following research questions (RQ):

- RQ1 Is PVC useful for informing users of the status of running programs?

⁴⁾Dynamically allocated memory can be visualized if *sizeof(type)* is used in arguments of *malloc*

Table 3: Comparison of PVC, SeeC, and no tool in terms of median, mean, and statistical test

		Number of correct answers	Time to answer [s]
Median (Mean)	no tool	3 (2.5)	1002.5 (1148.4)
	SeeC	2 (2.1)	1030 (1045.5)
	PVC	3 (3.1)	620.5 (623.3)
Steel-Dwass test (p-value)	PVC / no tool	<0.1	<0.01
	PVC / SeeC	<0.05	<0.05
	no tool / SeeC	n.s.	n.s.

- RQ2 Is PVC useful for novices who are learning C programming language concepts?
- RQ3 Is PVC as useful as similar tools for visualizing running program status, and is it more accessible as an application?

We conducted an experiment and a survey to evaluate PVC with respect to RQ1–3.

5.1 Experimental Setting

The participants were 30 undergraduate or graduate students majoring in computer science and engineering. Ten performed the tasks using PVC, 10 using SeeC⁵⁾, and remaining 10 did not use a visualization tool (viewed the code only and used a pen and paper). After completing the tasks, all participants used PVC and answered the questionnaire.

The experiment involved four tasks. The source code used in this experiment was the same as the code shown on our demo site⁶⁾. Table 2 shows the examination questions (T1–4) for each task and the questions (Q1–4) in the questionnaire. The participants completed the questionnaire upon finishing the tasks using PVC. Prior to the experiment, we explained PVC and provided a tutorial on how to use it. (The explanation was almost identical to that in Fig. 3.). During the experiment, we measured the time taken to answer each question and checked the answers. Q1 to Q3 were evaluated on a scale from 1 to 5, where 5 is the most positive.

5.2 Experimental Results

Figure 4 shows a box plot of the time taken to answer each task, while Fig. 5 shows a bar graph of the percentage of correct answers for the tasks. Figures 6 and 7 show the questionnaire responses. Figure 6 shows the participants' answers to Q1–3, which evaluate the usefulness and accessibility of a tool, whereas Fig. 7 shows the answers to Q4, which assesses the utility of PVC. In other words, Fig. 6 represents a measure of the usefulness and accessibility of the PVC program (Q1–3), while Fig. 7 relates the appropriateness of the tool to the application (Q4).

RQ1: Figures 4 and 5 clearly indicate that the group using PVC generally answered the questions more accurately and faster than the other groups. Table 3, which compares the mean values of these results, shows that the group using PVC answered the questions on average 1.8 times faster and gave 24% more correct responses than

⁵⁾For the above reason, we judged that there is no visualization difference between PT and SeeC. Hence, the experiment only employed SeeC. If PT was selected, T3 would change to a File I/O problem.

⁶⁾<http://play-visualizer-c.herokuapp.com>

Figure 4: Box plot of the time required to complete each task **Figure 5: Percentage of correct answers for T1–T4 (for RQ1)** **Figure 6: Results of Q1-3 (for RQ2 and RQ3)****Figure 7: Result of Q4 (for RQ3)**

the group working without a tool (no tool). Compared to the group working with SeeC, those working with PVC responded 1.7 times faster and provided 19% more correct answers, demonstrating that PVC is well suited to the tasks.

Figure 4 shows the results as a box plot. Table 3 also shows the *p*-values between the groups and evaluates them using the Steel-Dwass test [4]⁷⁾. A significant difference exists between the groups, except for the relationship between the SeeC and the no tool groups.

Focusing on the results of T3 and T4 of the SeeC group, the correct answer rates are worse than the no tool group, which was unexpected. Moreover, for T3, the SeeC group took more time on average to answer than the no tool group. We speculate that the

⁷⁾We selected the Steel-Dwass test, which is a non-parametric test, because it is difficult to determine whether the scores and times of the tasks follow a normal distribution that strictly avoids multiple test.

reasons for these are as follows: T3 — SeeC does not fully support dynamic memory allocation. Thus, participants become confused or have to switch to working with pen and paper, which is more time-consuming. T4 — PVC can visualize the depth of recursive call as a number, but SeeC cannot. This may lead to the difference in the correct answer rate.

Moreover, the result of Q1 (shown in Fig. 6) indicates that all participants felt that PVC is very useful for solving the problems in the experiment. (All participants responded with 4 or 5.) These results indicate that PVC can convey the status of running programs to programmers in general.

RQ2: About 90% of the participants feel that PVC is useful for novices learning some concepts of C programming language (Fig 6, Q2). As expected, most users felt that PVC is a useful aid for learning pointer and dynamic memory allocation (Fig. 7, Q4). Additionally, participants felt that the program is also useful for functions. Based on these results, we speculate that PVC is useful for checking the answer to T2, which is challenging as it involves a recursive function. Given that task T2 yielded the lowest percentage of correct answers, it is inferred that it is the most challenging part of the experiment.

RQ3: About 80% of the participants felt that PVC is more accessible than other existing visualization or debugging tools (Fig. 6, Q3).

Hence, the experiment confirms that PVC mostly satisfies RQ 1–3 proposed at the beginning of the study

5.3 Threats to Validity

All the participants in this study had basic knowledge of the C language and we provided a tutorial of PVC. However, the degree of proficiency varied within the groups. Some participants used PVC very well, while others struggled in the experiment. These attributes may affect the results of the experiment and pose a threat to internal validity.

Most participants are students of Waseda University. If we repeated this experiment with people belonging to another group or organization, the results may vary. The results may depend on demographic (e.g., age), programming skill level, and experience in the field, posing a threat to the experiment's external validity.

5.4 Limitations

Currently, PVC does not support minor language features, built-in functions, and compiler features. For example, it does not support *union* and external libraries. However, PVC is designed for novices. Consequently, we believe that these oversights are acceptable because these concepts are difficult for novices to comprehend. Moreover, PVC does not display compiler error messages because PVC is a visualization tool and not a compiler. In addition, PVC cannot hide or fold up figures, which may be confusing to users if there are numerous variables.

6 CONCLUSIONS AND FUTURE WORK

We proposed new visualization technique for C languages called PVC. It is a browser-based web application inspired by previous studies. The application is open-access and free to try at the listed

address (see Section I). The experiment reveals that PVC is useful not only for novices, but also programmers in general.

In the future, we plan to investigate whether PVC can help students learn programming by evaluating users' programming skills after using PVC. Moreover, we will continue to develop and improve PVC. For example, our visualize application currently supports only C language. It does not support the full range of C language syntax and standard library functions.

ACKNOWLEDGMENTS

This work was partially supported by JST Presto Grant Number JPMJPRI4D4.

REFERENCES

- [1] Herbert Baerten and Frank Van Reeth. 1998. Using VRML and JAVA to visualize 3D algorithms in computer graphics education. *Computer networks and ISDN systems* (1998).
- [2] Michelle Craig and Andrew Petersen. 2016. Student difficulties with pointer concepts in C. In *Proceedings of the Australasian Computer Science Week Multi-conference*. ACM.
- [3] Wim De Pauw, Erik Jensen, Nick Mitchell, Gary Sevitsky, John Vliissides, and Jeahua Yang. 2002. Visualizing the execution of Java programs. In *Software Visualization*. Springer.
- [4] MM Desu and Damaraju Raghavarao. 2003. *Nonparametric statistical methods for complete and censored data*. CRC Press.
- [5] Stephen H Edwards, Daniel S Tilden, and Anthony Allevato. 2014. Pythy: improving the introductory python programming experience. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM.
- [6] Matthew Heinsen Egan and Chris McDonald. 2014. Program visualization and explanation for novice C programmers. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*. ACS.
- [7] Micaela Esteves and AJ Mendes. 2003. OOP-Anim, a system to support learning of basic object oriented programming concepts. In *Proceedings of the CompSysTech*.
- [8] Licia Florio and Klaas Wierenga. 2005. Eduroam, providing mobility for roaming users. In *Proceedings of the EUNIS 2005 Conference, Manchester*.
- [9] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 579–584.
- [10] Philip J. Guo, Jeffery White, and Renan Zanelatto. 2015. Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (VL/HCC '15)*. 79–87. <https://doi.org/10.1109/VLHCC.2015.7357201>
- [11] Ryosuke Ishizue, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. 2017. An interactive Web Application Visualizing Memory Space for Novice C Programmers. In *Proceedings of the ACM SIGCSE*. ACM, 710–710.
- [12] Richardo Jiménez-Peris, Cristóbal Pareja-Flores, Marta Patiño-Martínez, and J Ángel Velázquez-Iturbide. 1997. The locker metaphor to teach dynamic memory. In *ACM SIGCSE Bulletin*. ACM.
- [13] Nobuya Koike and Kentaro Go. 2012. A Proposal of Interaction on Visualization Using Address Space Representation in Runtime Program. *Interaction* (2012).
- [14] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin*. ACM.
- [15] S-P Lahtinen, Erkki Sutinen, and Jorma Tarhio. 1998. Automated animation of algorithms with Eliot. *Journal of Visual Languages & Computing* 9, 3 (1998), 337–349.
- [16] Yoshiaki Matsuzawa, Kazunori Sakamoto, Takashi Ohata, and Katsuhiko Kakehi. 2015. A Programming Language Translation System for Programming Education. In *SIGCE*. IPSJ.
- [17] Iain Milne and Glenn Rowe. 2002. Difficulties in learning and teaching programming-views of students and tutors. *Education and Information technologies* (2002).
- [18] Iain Milne and Glenn Rowe. 2004. Ogre: Three-dimensional program visualization for novice programmers. *Education and Information Technologies* (2004).
- [19] Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. 2004. Visualizing programs with Jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces*. ACM, 373–376.
- [20] Linda Null and Karishma Rao. 2005. CAMERA: Introducing memory concepts via visualization. In *ACM SIGCSE Bulletin*. ACM.
- [21] Jaishankar Sundararaman and Godmar Back. 2008. HDPV: interactive, faithful, in-vivo runtime state visualization for C/C++ and Java. In *Proceedings of the 4th ACM symposium on Software visualization*. ACM.