2018-07-31

Overview of using visualisation in programming learning

Alhammad, S

http://hdl.handle.net/10026.1/12492

International Journal of Advances in Electronics and Computer Science
IRAJ

International Journal of Advances in Electronics and Computer Science, ISSN(p): 2394-2835 Volume-5, Issue-7, Jul.-2018
http://iraj.in

OVERVIEW OF USING VISUALISATION IN PROGRAMMING LEARNING

[1]SARAH ALHAMMAD, [2]SHIRLEY ATKINSON, [3]LIZ STUART

[1,2,3] School of Computing, Electronics and Mathematics, Plymouth University, Plymouth, United Kingdom
[1,2]Collage of Computer Sciences, Princess Nora bint Abdulrahman University, Riyadh, Saudi Arabia

E-mail: [1]sarah.alhammad@plymouth.ac.uk, [2]shirley.atkinson@plymouth.ac.uk, [3]L.stuart@plymouth.ac.uk

Abstract - There is a high demand for mechanisms that support programming teaching, particularly in finding a solution to the bottleneck in programming education. Nowadays, different methods of teaching can support the learning process and motivate students to learn. These methods improve the thinking and creativity that lead to defining and analysing the problem of supporting programming learning to devise ideal solutions. Approaches such as visualising the code or using a memory diagram to trace the program's execution have made a vital contribution to the process of teaching and learning how to program.

The purpose of this paper is to provide an overview of studies that have been conducted in visualisation to support programming learning. Moreover, tools that follow the visualisation and memory-referencing approach will be investigated in this paper.

Index terms - Memory diagram, Memory Transfer Language, Programming learning, Visualisation.

I. INTRODUCTION

Memory reference visualisation, or memory transfer language (MTL), is defined in [1] as a language or device used by programmers to describe the impact of lines of code on computer memory (RAM). Visualising the impact of each line of code on memory allows novices to grasp the purpose and impact of each instruction. In this way, students' comprehension will improve because they can predict the result during the execution time. A memory diagram as a pedagogical tool is carefully designed to aid students' understanding of programming. The use of a memory diagram has a direct relationship between code and its effect. It does not require a student to learn any concepts, compared to a flowchart, which requires students to know the meanings of symbols and their connections. A memory diagram is a portable, flexible and scalable tool, as it is machine- and language-independent. It could also be used as a code design and testing tool [1],[2].

This paper will provide a literature review to contribute to the use of visualisation and memory diagrams in programming learning. Moreover, the paper will present studies that have been conducted to evaluate the visualisation method. In the final part of the paper, an overview of the most common tools that rely on visualisation will be presented to show their effects on students' comprehension.

II. OVERVIEW OF VISULISATION METHOD

Many researchers have pointed out the advantages of using visualisation in programming learning. In [2], the researchers tested hypotheses regarding the use of a memory diagram in teaching programming to enhance students' ability to write code. A class experiment was conducted on 100 students, who were divided into two groups (control and experimental groups) and whose test scores in the final exam were compared. There was a significant difference in the test scores of the two groups (the average score was 64.67% for the experimental group and 60.02% for the control group), so the hypothesis was accepted at a certain level.

Mselleand Twaakyondo in [2] also considered the impact of using MTL on reducing misunderstandings in programming teaching. An experiment using MTL to teach programming used error counts in the exam for two groups of students, one of which used MTL and the other a conventional approach. The statistical numbers indicate that the number of errors made by the MTL group (208 total errors) was less than the number of errors made by the control group (392 total errors).

The study in [3] found that a visualisation environment contributes to increasing apprehension and reducing the effort and time consumed during programming lectures. The visualisation also overcomes the barriers to programming, such as mechanical and sociological barriers.

The researchers in [4] evaluated the "Turtlet" tool as an approach to using visualisation in teaching programming. Two surveys were distributed to and completed by students at the end of the course. The studies measured aspects such as complexity, interest and ease of use. The first survey was analysed to evaluate students' opinions of the tool. The results showed that 35% of students liked the tool, 20% did not like it and the rest were neutral. The second survey analysed the demonstration and course exercise used by the tool. The results collected from the second study proved that 81% of the students preferred taking the course project with Turtlet, 96% enjoyed the exercise that was created using Turtlet

and 85% preferred the demonstration. Overall, comparing students from 2005 to 2007, there was a decrease in the rate of course dropout, withdrawal or failure. The rate was 55.1% and 27.3% before and after Turtlet, respectively.
BackStop is another tool that uses visualisation in teaching programming. In [5], the researchers measured the usefulness of BackStopin relation to two aspects: the time needed to find errors and solve the errors in a given task (debugging a logical error). Two experiments were conducted: the first recorded the time required by the students to identify and fix the logical errors related to a given task, while the second was conducted to find logical errors in the program. The results of the first research found that 76% of the total students were able to recognize the mistakes and fix them within eight minutes. On the other hand, the students in the control group who were selected from the top students were able to correct the errors in less than five minutes. However, the reason it took so long was that the messages in BackStop were too long, and it took time to read them. In the second experiment, 47% students were able to find the errors. The students claimed that the debug tool (BackStop) helped them find the errors.
Hagan and Markham investigated the use of BlueJ as a visualisation tool to collect students' opinions of and attitudes towards BlueJ. The focus of a study conducted by Hagan and Markham [6] was not on syntax, but on the full picture of programming. The experiment was conducted at Monash University with 350 students who started to write a single class and then gradually began to write more classes. The students used the interactive visual environment of BlueJ, so they advanced from that environment to learn the object-oriented concept. During the semester, a series of surveys were distributed to evaluate the students' expectations and their performance in BlueJ. The study collected information about the students and their background knowledge in programming, and it then evaluated their BlueJ experiences. The survey showed that the majority of students had a positive shift in their relation to BlueJ. However, they still doubted its stability and reliability [6].
An example of using BlueJ as a pedagogical tool is implementing a project-building clock. The students should divide the problem into sub-problems; in our case, they have to implement two classes: DisplayNumber and DisplayClock. The DisplayNumber class displays a time consisting of two values, hours and minutes. On the other hand, the DisplayClock class is used to describe the clock. The solution steps can be implemented in the BlueJ platform by drawing the classes and modifying them at any time. Students benefit from the tool by learning the concept of object-oriented programming using the interactive mode [7].
The effectiveness of using Alice as a visual tool for learning was examined by comparing Alice to other platforms, such as SCRIBBLER, Microsoft Robotic Developer Studio 2008R3, NXT Tech Virtual Robotic Worlds and Lego NXT2.0.An experiment was conducted in Icesi University involving 15 lessonscovering the basics of programming. Each lesson containing video phones, and a workshop , the videos included tutorials covering specific topics with three parts: a theoretical introduction, a step-by-step exercise and results. The result of their experience suggested that using Alice raised the students' interest in programming. A questionnaire was distributed at a workshop held after the experiments to examine the effectiveness of using the tool. The results showed that students held a negative view of programming languages in general before the experiment, where 58% of students found them hard to understand, 92% said they had enough experience with the lessons and about 67% said they had fair or good experiences with programming languages, such as Java and C++. The result suggested that using Alice raised the students' interest in programming. Alice helps strengthen the theoretical concepts, while ROBOTICS and Lego help students to observe their code becoming an action[8].
The study in [9] evaluated and tested the usability of the Memview tool as a visual debugger in a classroom experiment, as Memview uses a picture of the memory. The authors found that the tool reduces students' misunderstanding, as well as the effort and length of lectures.
AnimPascal has been evaluated as a visual tool used in programming learning. A typical laboratory class can be observed while students solve binary search algorithms using AnimPascal. The recording features in AnimPascal gave the instructors an overall idea of students' errors, which will consequently help them to focus on these errors to avoid the misunderstanding. The explicit messages in case of failure assisted the students in problem-solving[10].
Does a memory diagram help students to understand code execution? This question was raised in [11] to evaluate the use of a memory diagram by using the Code Memory Diagram (CMD) tool when used by (developer/author). A survey consisting of a questionnaire, video observation and participants' (students') observations was distributed for the evaluation. Of 25 students in total, the questionnaire and participant observation showed an increase in understanding among 52% of students, while the remaining 48% showed no change in understanding. After that, the researcher tested the usability of CMD during the teaching session. It revealed that the software enhanced students' comprehension of structured programming during the teaching session[12].

III. VISUAL TOOLS FOR SUPPORTING PROGRAMMING LEARNING

This section gives an overview of some tools that have been used in programming learning in academic institutions to increase comprehension among students or novice programmers. The tools rely on the concept of visualising or animating the program execution.

A. The BlueJ Tool

eThe first use of BlueJ to teach Java was in 1999 in a computer science introductory course. The BlueJ tool was implemented based on the Unified Modeling Language (UML) and Java language[13]. It uses a graphical representation to show classes and objects within a project (Fig.1). Students can create an object thought a Graphical User Interface (GUI) without writing codes, and they can make this object interact with other objects, which can help the students strengthen their understanding of the concepts of object-oriented programming[14].

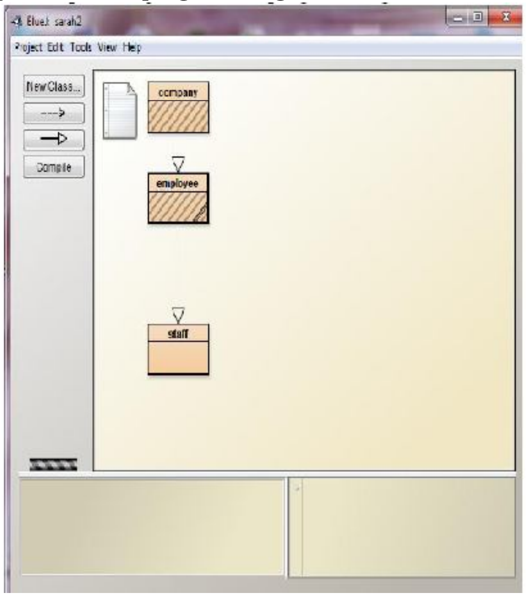
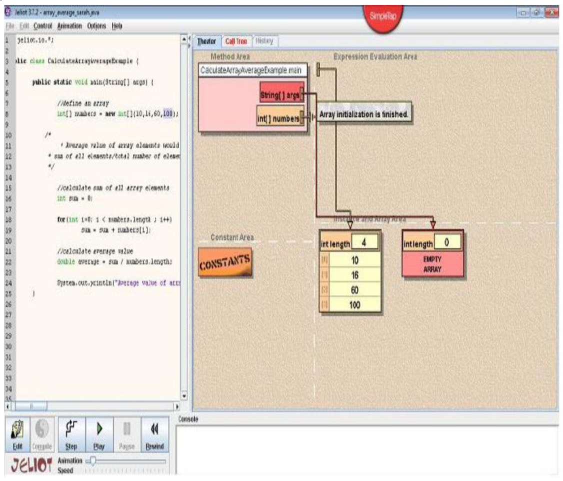


Figure 1: BlueJ

B. The Jeliot3 Tool

The visualisation of object-oriented programs has been developed with Jeliot3 after several generations. The first generation was Eliot, which was designed to produce algorithm animations. After that, JeliotIwas specially designed for Internet use, and Jeliot 2000 was dedicated to novice programmers. What makes Jeliot3 different from the previous generations is the extension of visualisingobject-oriented concepts. Jeliot3 differs from BlueJ in its provision of dynamic visualisation, which is absent from BlueJ. In Jeliot3, many features have been added to suit user requirements. These functions involve ease of use, consistency ,continuous and complete visualisation that is extensible both internally and externally. The visualised components that show up in a separate window of Jeliot3 are designed to simplify use of the tool with an animation frame structure (Fig.2). Error explanation and the ability to highlight the line of error is also provided in Jeliot3, as it uses UML notations to represent the objects and their relations to clarify object-oriented concepts for students [15].


Jeliot 3.7.2 - array_average_sarah.java
File Edit Control Animation Options Help
Theater Call Tree History
Method Area
Expression Evaluation Area
CalculateArrayAverageExample.main
String[] args
int[] numbers
Array initialization is finished.
Constant Area
CONSTANTS
Instance and Array Area
int length 4
10
16
60
100
int length 0
EMPTY ARRAY
Console
Edit Compile Step Play Pause Rewind
Animation Speed

```
jeliot.io.*;

lic class CalculateArrayAverageExample {

    public static void main(String[] args) {

        //define an array
        int[] numbers = new int[]{10,16,60,100};

      /*
         * Average value of array elements would
       * sum of all elements/total number of eleme
       */

        //calculate sum of all array elements
        int sum = 0;

        for(int i=0; i < numbers.length ; i++)
            sum = sum + numbers[i];

        //calculate average value
        double average = sum / numbers.length;

        System.out.println("Average value of arr
    }
```

Figure 2: Jeliot3

C. The Alice Tool

The Alice tool uses a built-in drag-and-drop interface to program a 3D world with interaction (Fig.3). Alice is a project of Carnegie Mellon University in the United States, and it is designed for young people to introduce them to programming. They learn the fundamentals of object-oriented programming through the graphical representation of objects[8].

D. WebTasks

WebTasks is a programming task database tool that runs entirely in a web browser, and it does not require a program to be downloaded (even SDK on the students' devices). WebTasks contains JSP pages that run on the Apache Tomcat Server (Fig. 4).
The students pick an assignment and then write the body of the methods; most of the methods have a predefined header. The code is then tested using JUnit. The series of testing continues as recursion until the problem has been solved. The Department of Computer Science in Technology at Darmstadt University (Germany) developed a system using WebTasks to solve about 118 Java programming

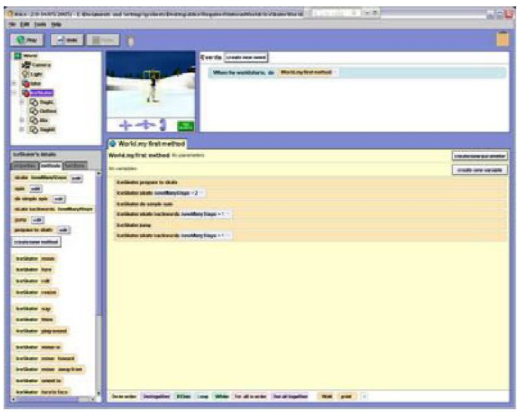
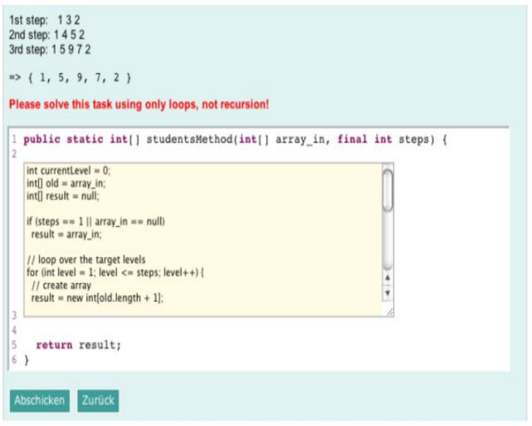


Figure 2: The Alice Tool

tasks. The computer science students can log into the system and try all its tasks, and this encourages them to write Java programs, submit them and receive fast feedback on corrections [16].



```
1st step:   1 3 2
2nd step: 1 4 5 2
3rd step: 1 5 9 7 2

=> { 1, 5, 9, 7, 2 }
```

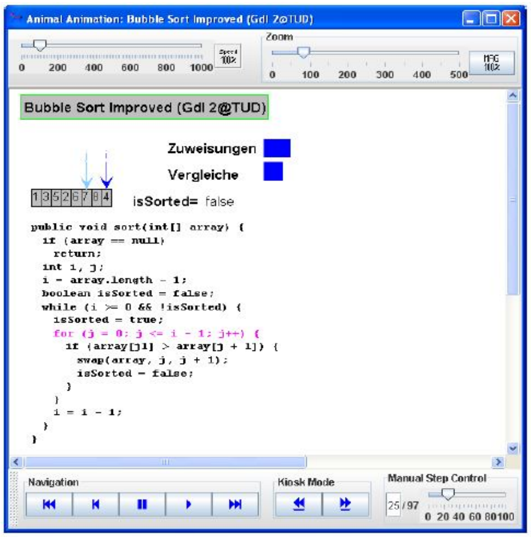
Please solve this task using only loops, not recursion!

```
public static int[] studentsMethod(int[] array_in, final int steps) {

int currentLevel = 0;
int[] old = array_in;
int[] result = null;

if (steps == 1 || array_in == null)
  result = array_in;

// loop over the target levels
for (int level = 1; level <= steps; level++) {
  // create array
  result = new int[old.length + 1];


  return result;
}
```

Figure 3: WebTasks

E. The ANIMAL System

Visualising and animating algorithm methods are used for solving more complicated problems in data structures, such as binary trees and graphs, as well as for sorting and searching for algorithms (Fig.5). The process helps students to understand the behaviour of these structures and algorithms. The ANIMAL system is based on this method, and it follows the concept of visualising the representation of source codes and highlighting the current executing line [16].

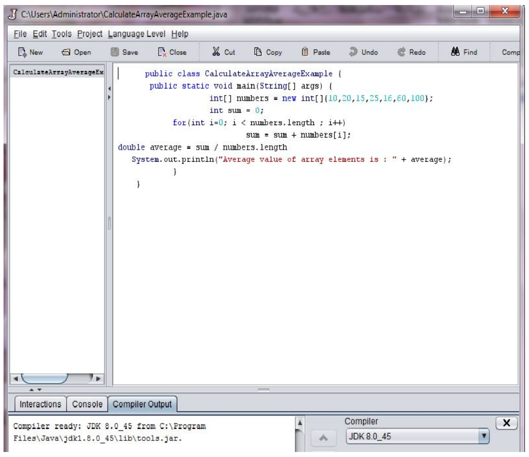


```
public void sort(int[] array) {
  if (array == null)
    return;
  int i, j;
  i = array.length - 1;
  boolean isSorted = false;
  while (i >= 0 && !isSorted) {
    isSorted = true;
    for (j = 0; j <= i - 1; j++) {
      if (array[j1] > array[j + 1]) {
        swap(array, j, j + 1);
        isSorted = false;
      }
    }
    i = i - 1;
  }
}
```

Figure 4: The ANIMAL System

F. DrJava

The purpose of the DrJava tool is to teach students how to design programs in Java, as well as how to test and debug the program. It consists of a window with two panes linked by an integrated compiler (Fig.6). The interaction pane is used to input Java expressions, and the definition pane is used to enter and edit class definitions. The features in DrJava include the interaction window, which has a 'read-eval-print loop' (REPL) to enable access to program components without recompiling. Testing and debugging features are also available using REPL to test the methods individually. Moreover, students can debug the code without the need to learn the debugging mechanisms. DrJava includes an editor to detect syntax errors; it can highlight the parenthesis. DrJava also has an integrated compiler bundled with the Java compiler [17].



C:\Users\Administrator\CalculateArrayAverageExample.java
File Edit Tools Project Language Level Help
New Open Save Close Cut Copy Paste Undo Redo Find Comp
CalculateArrayAverageEx
Interactions Console Compiler Output
Compiler
JDK 8.0_45

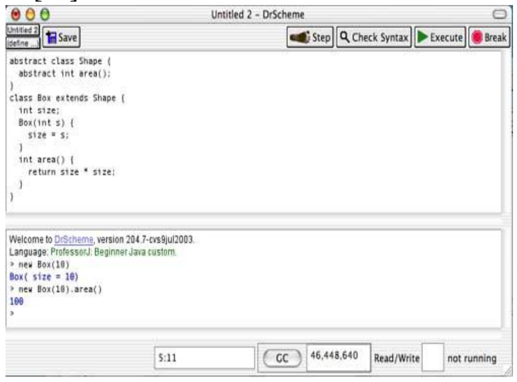
```
    public class CalculateArrayAverageExample {
     public static void main(String[] args) {
                int[] numbers = new int[]{10,20,15,25,16,60,100};
                int sum = 0;
           for(int i=0; i < numbers.length ; i++)
                     sum = sum + numbers[i];
double average = sum / numbers.length
  System.out.println("Average value of array elements is : " + average);
           }
    }
```

```
Compiler ready: JDK 8.0_45 from C:\Program
Files\Java\jdk1.8.0_45\lib\tools.jar.
```

Figure 5: DrJava

G. ProfessorJ

ProfessorJ is a pedagogical environment that presents an interface for the Java compiler (Fig.7). The ProfessorJ interface consists of two windows: a definition window containing the code and an interaction window that provides a REPL to experiment with the code. It contains three levels of difficulties: beginner, intermediate and advanced. In the beginner mode, students can define the declaration construction and its restriction. The intermediatemode starts to teach object-oriented programming. The advanced mode introduces loops and arrays. The code in ProfessorJ highlights the keywords and variables, and it contains a check syntax tool. The students can track their variables by a binding instant of the variable to all of its uses using arrows. ProfessorJ has a feature that highlights errors outside of the debugging environment, and that can stop the execution at any time during the debugging mode [18].



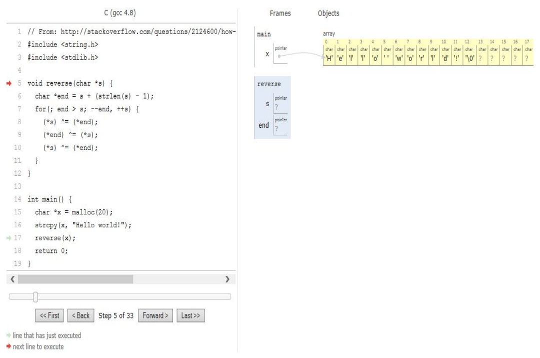
```
abstract class Shape {
  abstract int area();
}
class Box extends Shape {
  int size;
  Box(int s) {
    size = s;
  }
  int area() {
    return size * size;
  }
}
```

```
Welcome to DrScheme, version 204.7-cvs9jul2003.
Language: ProfessorJ: Beginner Java custom.
> new Box(10)
Box( size = 10)
> new Box(10).area()
100
>
```

Figure 6: ProfessorJ

H. Online Python Tutor

Online Python Tutor is a web-based programming tool that uses visualisation extensively. This open-source software enables users to embed their code into a web page, as shown on the left in(Fig.8). Subsequently, the code can be traced using navigation buttons. The visualisation of the codeis shown on the right in the figure. This visualisation enables the user to watch the dynamic execution of the program. As the program executes, it depicts changes to frames and objects. Additionally, it has a program output area. The tool provides explanations of errors, with the indicators pointing to the line on which the error occurred [19].



```
// From: http://stackoverflow.com/questions/2124600/how-
#include <string.h>
#include <stdlib.h>

void reverse(char *s) {
  char *end = s + (strlen(s) - 1);
  for(; end > s; --end, ++s) {
    (*s) ^= (*end);
    (*end) ^= (*s);
    (*s) ^= (*end);
  }
}

int main() {
  char *x = malloc(20);
  strcpy(x, "Hello world!");
  reverse(x);
  return 0;
}
```

Figure 7: Online Python Tutor

I. Visual Logic

Visual Logic uses the concept of iconic programming (icons and flowcharts) to visualise the program. Visual Logic has no code to be written (Fig.9). Instead, the user creates a flowchart representing the code. Subsequently, the tool traces the flow of the code. The tool demonstrates the outcomes of executing each icon in the flowchart in popup windows. Visual Logic does not support object-oriented programming[20].

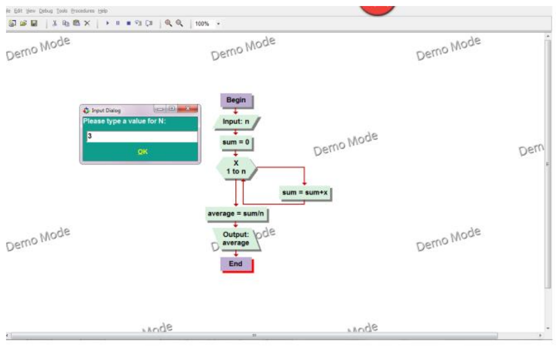


Figure 8: Visual Logic

CONCLUSION

Visualisation tools have been introduced for learning programming by many academic institutions. Some of these tools achieve the goal of and make tangible contributions to learning programming, at least from tool developers' perspectives. However, visualisation tools need to be evaluated continuously to reach their maximum benefits.

REFERENCES

[1] L. J. Mselle, "Enhancing Comprehension by Using Random Access Memory (RAM) Diagrams in Teaching Programming : Class Experiment," Sci. Technol., 1989.
[2] L. J. Mselle and H. Twaakyondo, "The impact of Memory Transfer Language (MTL) on reducing misconceptions in teaching programming to novices," Int. J. Mach. Learn. Appl., vol. 1, pp. 1–6, 2012.
[3] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," ACM Comput. Surv., vol. 37, pp. 83–137, 2005.
[4] J. Kasurinen, M. Purmonen, and U. Nikula, "A Study of Visualization in Introductory Programming," Ppig '08, no. Winslow 1996, pp. 181–194, 2008.
[5] C. Murphy, E. Kim, G. Kaiser, and A. Cannon, "Backstop: a tool for debugging runtime errors," ACM SIGCSE Bull., vol. 40, no. 1, p. 173, 2008.
[6] D. Hagan and S. Markham, "Teaching Java with the BlueJ environment," Proc. Australas. Soc. Comput. Learn. Tert. Educ. Conf. ASCILITE 2000, 2000.
[7] B. Sun, "Java teaching based on BlueJ platform," 2nd Int. Conf. Inf. Eng. Comput. Sci. - Proceedings, ICIECS 2010, pp. 2–5, 2010.
[8] S. L. Salcedo and A. M. O. Idrobo, "New tools and methodologies for programming languages learning using the scribbler robot and Alice," Proc. - Front. Educ. Conf. FIE, pp. 1–6, 2011.
[9] P. Gries, V. Mnih, J. Taylor, G. Wilson, and L. Zamparo, "Memview: A Pedagogically-Motivated Visual Debugger," Proc. Front. Educ. 35th Annu. Conf., pp. S1J–11–S1J–16, 2005.
[10] M. Satratzemi, V. Dagdilelis, and G. Evagelidis, "A system for program visualization and problem-solving path assessment of novice programmers," ACM SIGCSE Bull., vol. 33, no. 3, pp. 137–140, 2001.
[11] M. Dixon, "Code-Memory Diagram Animation Software Tool : Towards on-Line Use," proceeding IASTED Int. Conf. WEB-BASED Educ., no. February 16–18,2004,Innsbruck,Austria, pp. 601–603, 2004.
[12] M. Dixon, "year long 3-user trial of code-memory diagram animation software for teaching computer programming : learning ,object orientedprogramming ,& workloads" in International conference in Computers and Advanced Technology in Education, 2005, no. phase 3, pp. 238–243.
[13] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg, "The BlueJ system and its pedagogy," Comput. Sci. Educ., vol. 13, no. 4, pp. 1–12, 2003.
[14] J. Bennedsen and C. Schulte, "BlueJ Visual Debugger for Learning the Execution of Object-Oriented Programs?," ACM Trans. Comput. Educ., vol. 10, no. 2, pp. 1–22, 2010.
[15] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, "Visualizing programs with Jeliot 3," Proc. Work. Conf. Adv. Vis. interfaces - AVI '04, p. 373, 2004.
[16] G. Rößling, "A family of tools for supporting the learning of programming," Algorithms, vol. 3, pp. 168–182, 2010.
[17] E. Allen, R. Cartwright, and B. Stoler, "DrJava: A lightweight pedagogic environment for Java," ACM SIGCSE Bull., vol. 34, pp. 137–141, 2002.
[18] K. Gray and M. Flatt, "ProfessorJ: a gradual introduction to Java through language levels," Companion 18th Annu. ACM SIGPLAN …, pp. 170–177, 2003.
[19] P. J. Guo, "Online python tutor: Embeddable web-based program visualization for cs education," SIGCSE 2013 - Proc. 44th ACM Tech. Symp. Comput. Sci. Educ., pp. 579–584, 2013.
[20] D. Gudmundsen and L. Olivieri, "Using Visual Logic © : Three Different Approaches," J. Comput. Sci. Coll., vol. 26, no. 6, pp. 23–29, 2011.

★★★