Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 178 (2007) 43–49
www.elsevier.com/locate/entcs

Automatic Generation of Prediction Questions during Program Visualization

Niko Myller[1,2]

Department of Computer Science
University of Joensuu
Joensuu, Finland

Abstract

Based on previous research, it seems that the activities performed by and the engagement of the students matter more than the content of the visualization. One way to engage students to interact with a visualization is to present them with prediction questions. This has been shown to be beneficial for learning. Based on the engagement taxonomy and benefits of the question answering during the algorithm visualization, we propose to implement an automatic question generation into a program visualization tool, Jeliot 3. In this paper, we explain how the automatic question generation can be incorporated into the current design of Jeliot 3. In addition, we provide various example questions that could be automatically generated based on the data obtained during the visualization process.

Keywords: Program visualization, Engagement taxonomy, Automatic question generation.

1 Introduction

According to Hundhausen et al. [6], the activities performed by and the engagement of the students matter more than the content of the visualization. Thus, a research program has been laid down in which the level of engagement (*engagement taxonomy*) and its effects on learning with algorithm or program visualization are being studied [13]. One of the ways to engage students to interact with a visualization is to present them with questions, which ask the students to predict what happens next in the execution or visualization (level 3: responding) [12]. This has been shown to be beneficial for learning as well [3,11]. Furthermore, interaction and question answering during learning have been found to have a positive influence on problem-solving ability in the given domain [5]. In addition to the benefits

[1] I want to thank Erkki Sutinen and Moti Ben-Ari for their helpful comments and guidance during the research work, and Andrés Moreno and Roman Bednarik for fruitful discussions related to the topic of this paper. Part of this work was carried out during author's visit to Massey University, Palmerston North, New Zealand.

[2] Email: firstname.lastname@cs.joensuu.fi

doi:10.1016/j.entcs.2007.01.034

found in the literature, in my work in progress, I am trying to show that actually the engagement taxonomy has a linkage to collaboration. The hypothesis in this research is that the higher the engagement level the larger the positive impact on collaboration.

Based on the found benefits of the question answering during visualization, we propose to implement an automatic question generation into a program visualization tool, *Jeliot 3* [9]. In this paper, we explain how the automatic question generation can be incorporated into the current design of Jeliot 3. In addition, we provide various example questions that could be automatically generated based on the data obtained during the visualization process. Finally, conclusions and future directions are presented.

2 Jeliot 3

Jeliot 3 is a program visualization system that visualizes the execution of Java programs [9]. It has been designed to support the teaching and leaning of introductory programming. Jeliot visualizes the data and control flow of the program. In a classroom study, it was found that especially the mid-performers benefited from the use of Jeliot while the performance of others was not harmed [1].

We describe the structure of Jeliot 3 in Figure 1 in order to explain in the next section how the automatic question generation fit into the current design. The user interacts with the user interface and creates the source code of the program (1). The source code is parsed and checks are performed before the actual interpretation by *DynamicJava*, a Java interpreter (2). During the interpretation, a representation of the program's execution, MCode, is extracted (3). MCode is assembly like language in which each line represents a single instruction that contains instruction type, instruction identifier (can be used as a reference), variable number of operands (references to other instruction ids, type and value) and the instruction's location in source code [8]. MCode is interpreted and the directions are given to the visualization engine (4 and 5). The user can control the animation by playing, pausing, rewinding or playing step-by-step the animation (6). Furthermore, the user can give input to the program executed by the interpreter (6, 7 and 8). For further information related to the internal structure of Jeliot the reader is pointed to [8,10].

3 Automatic Question Generation in Jeliot 3

The steps of question generation are: 1) information collection, 2) question formation, and 3) its presentation during the animation. In order to generate prediction questions, we need to collect information related to the program interpretation. We list here items that are needed to generate a question:

- Type of Expression, so that different question text is generated for assignment expression compared to method call.
- Instruction identifier to ensure that the question is popped up at right time.

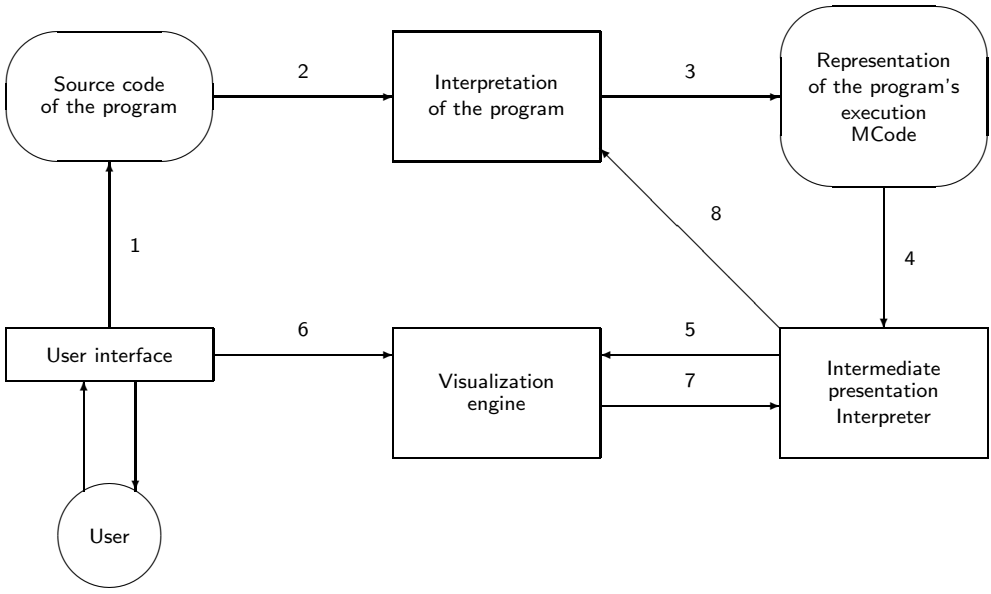

Fig. 1. The functional structure of Jeliot 3 [9].

- All the different concepts related to the expression that the question concerns. In this way, students performance in questions related to different concepts can be recorded into a user model.
- Correct answer that is the result of the program's interpretation.
- Zero to three incorrect answers depending on the question type: multiple choice questions have four answers, yes-or-no questions have two answers and open-ended questions have none. Currently, incorrect answers are determined randomly, but it could be possible to apply certain heuristics based on the previous steps in the execution and current values of variables.
- Location of the expression in the source code so we can highlight that part of the code when the question is shown.

As discussed in the previous section, the interpretation of the program produces a program trace that is called MCode. In order to collect the listed information, we implemented a *preprocessor for MCode*. It goes through the MCode before it is interpreted and extracts the information from the MCode. This information is saved so that MCode interpreter can query them based on the expression identifier during the interpretation. In the Figure 1, this phase would be located in the middle of the arrow 4.

Then during the interpretation of every MCode instruction, interpreter checks whether a question for the current instruction identifier is found. If a question is found, it is shown to the user before the instruction is animated. We adapted the *avInteraction* package developed by Rößling and Häussge [14] in order to present questions and collect users' answers. The answers can be saved into a file or a

database. Thus, this feature can be used in order to adapt the program visualization as well as to summatively evaluate the students performance as part of their course work or on-line exam.

As a proof of concept, we have implemented a restricted question generation that only asks questions related to the results of the assignment statements. An example of the generated question is displayed in Figure 2. The question is shown in the right together with the visualization and the related code segment is highlighted in the left.

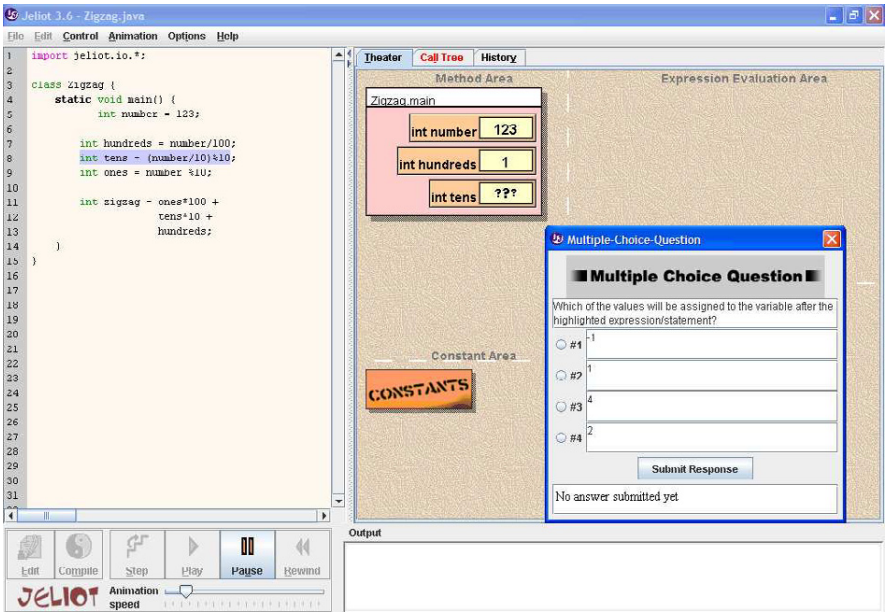

Fig. 2. An example question generated by Jeliot 3.

Moreover, different kinds of question related to the execution and animation of the program can be automatically generated. We list here some possibilities:

- Predict the result of any expression evaluation.
- Ask the user to click on the variable which is part of the expression, or into which the result of an expression evaluation is going to be assigned.
- In loops, it is possible to ask if the execution will continue for next round or not and in conditional statements, it is possible to ask if the execution will continue into the `then` or `else` part of the statement.
- User can be asked to click the line (or line number) that is being executed next, for instance, after a method call or in the beginning of a loop or an if-statement.
- In sorting, it would be possible to determine a swap operation and ask the user to click on those array cells that are going to be swapped.

It is not feasible to pop up questions in every possible place, because students can get annoyed or tired of the questions. Thus, there should be ways to determine when it is most appropriate and meaningful to generate a question. For example, Jeliot

could allow the user to select the variables or expression types that should generate questions and thus focus the questions on the selected concepts or parts of the program. Similarly to related systems (e.g. Problets and WadeIn (see Section 4)), we could adapt the question generation, visualizations and explanations based on the performance data of the user. We have done preliminary work on this direction, and it is described in [7].

Moreover, there should be a possibility for a teacher to manually create questions for programs in order to allow the use of question generation for quizzes or in-class exams. This can be achieved with the package that we use to display and save the question information, because it provides a file format for manual question specification [14].

4 Previous Work

Kumar et al. [4] have developed a system, called *Problets*, that generates exercises related to programming concepts (e.g. loops, pointers etc.) from language independent templates, thus supporting multiple programming languages. These exercises present a program and ask the user to identify the lines that generate output and determine what is the output during the execution of the program. In exercises regarding pointers, user needs to identify the code lines that are either syntactically or semantically erroneous. These exercises are delivered in the form of an applet that is connected to a server that handles the exercise generation and stores information related to the performance of the user. This is done in order to analyze what kind of exercises to present to the user.

When compared to the question generation in Jeliot 3, we can identify certain similarities and differences. Both ask questions related to the execution of a program. However, Problets are related to the program code, whereas questions in Jeliot can be related to the program code and visualization. This can give more variation in the question types as seen in Section 3. Jeliot supports dynamic aspects of the program execution, for example, user can give input to the program and the questions are adjusted accordingly because they are based on the information acquired during the interpretation process. Currently, Jeliot supports only Java. However, if interpreters for other programming languages are integrated into it, the question generation is language independent. Problets support multiple programming languages because of the language independent templates that are translated to the programming language in question. Problets can be used for learning and testing similarly as the automatic question generation in Jeliot 3.

Another related system is JHAVÉ [11] which combines visualization tools and support for interaction. It provides textual materials, questions, and other exercises related to the visualization, and thus engages users to the visualization. JHAVÉ supports only post-mortem visualization of the programs meaning that user needs to provide input data before the program or algorithm is run. The questions need to be defined manually into the source code of the program. These issues make the approach different from Jeliot.

WadeIn II [2] visualizes the expression evaluation in C language. The system consist of two modes: exploration and knowledge evaluation. The question generation is related to the knowledge evaluation mode in which student needs to demonstrate the understanding of the expression evaluation by simulating it. The task is to simulate the evaluation of the expression, whereas in Jeliot, a user is asked to predict what will happen next in the given context of the program and its execution.

5 Conclusion and Future Work

We have presented a way to automatically generate prediction questions during a program visualization automatically and a proof of concept implementation of it. We have also presented different types of questions that could be automatically generated with the same framework and ways to determine when those questions should be raised in order to support different ways of learning and testing.

As future work, we implement the proposed question types and test their usability. We also plan to study the use of question answering both during individual as well as collaborative learning of programming concepts and programming. We will variate the level of engagement to analyze its effects to the learning and collaboration. Furthermore, we can test how different types of questions support the understanding of programs and programming learning. For example, should the questions be related to data flow or control flow, or both. In addition to this, we are planning to use the feature in distance education as a part of the summative evaluation.

References

[1] Ben-Bassat Levy, R., M. Ben-Ari and P. A. Uronen, *The Jeliot 2000 program animation system*, Computers & Education **40** (2003), pp. 1–15.

[2] Brusilovsky, P. and T. D. Loboda, *WADEIn II: A Case for Adaptive Explanatory Visualization*, in: *Proceedings of The Eleventh Annual Conference on Innovation and Technology in Computer Science Education*, Bologna, Italy, 2006.

[3] Byrne, M. D., R. Catrambone and J. T. Stasko, *Evaluating animations as student aids in learning computer algorithms*, Computers & Education **33** (1999), pp. 253–278.

[4] Dancik, G. and A. Kumar, *A Tutor for Counter-Controlled Loop Concepts and Its Evaluation*, in: *Proceedings of Frontiers in Education Conference (FIE 2003)*, Boulder, CO, USA, 2003, pp. T3C–7–12.

[5] Evans, C. and N. J. Gibbons, *The interactivity effect in multimedia learning* (2006), accepted to Computers & Education.

[6] Hundhausen, C. D., S. A. Douglas and J. T. Stasko, *A meta-study of algorithm visualization effectiveness*, Journal of Visual Languages and Computing **13** (2002), pp. 259–290.

[7] Lin, T., A. Moreno, N. Myller, Kinshuk and E. Sutinen, *Inductive Reasoning and Programming Visualization, an Experiment Proposal*, in: *Proceedings of the Fourth International Program Visualization Workshop*, Florence, Italy, 2006, pp. 83–88.

[8] Moreno, A., "The Design and Implementation of Intermediate Codes for Software Visualization," Master's thesis, Deparment of Computer Science, University of Joensuu (2005), (http://cs.joensuu.fi/jeliot/files/Andres_thesis.pdf) (Accessed (5.9.2006)).

[9] Moreno, A., N. Myller, E. Sutinen and M. Ben-Ari, *Visualizing Program with Jeliot 3*, in: *Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI 2004*, Gallipoli (Lecce), Italy, 2004, pp. 373–380.

[10] Myller, N., "The Fundamental Design Issues of Jeliot 3," Master's thesis, Deparment of Computer Science, University of Joensuu (2004), (http://cs.joensuu.fi/jeliot/files/niko_thesis.pdf) (Accessed (5.9.2006)).

[11] Naps, T. L., *JHAVÉ – Addressing the Need to Support Algorithm Visualization with Tools for Active Engagement*, IEEE Computer Graphics and Applications **25** (2005), pp. 49–55.

[12] Naps, T. L., J. R. Eagan and L. L. Norton, *JHAVÉan environment to actively engage students in Web-based algorithm visualizations*, in: *Proceedings of the thirty-first SIGCSE technical symposium on Computer Science Education* (2000), pp. 109–113.

[13] Naps, T. L., G. Rösling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. Á. Velázquez-Iturbide, *Exploring the Role of Visualization and Engagement in Computer Science Education*, in: *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (2002), pp. 131–152.

[14] Rößling, G. and G. Häussge, *Towards Tool-Independent Interaction Support*, in: A. Korhonen, editor, *Proceedings of the Third International Program Visualization Workshop*, Warwick, England, 2004, pp. 110–117.