Mini-languages for non-Computer Science Majors: What are the Benefits?

Peter Brusilovsky, Olena Shcherbinina, Sergey Sosnovsky
School of Information Sciences, University of Pittsburgh
135 North Bellefield Avenue
Pittsburgh, PA 15260
Phone: 412 624 9404, Fax: 412 624 2788
E-mail: {peterb, ols1, sas15} @ pitt.edu

Abstract: Mini-languages for teaching principles of programming - such as Karel the Robot - were once used in top computer science departments to provide a "gentle introduction" to programming for computer science majors. The paper builds a case for the use of mini-languages in the context of introductory programming courses for non-computer science majors. We present a study that explored the use of Karel to teach introductory programming for information science majors.

Keywords: Mini-language, introductory programming, learning environment, information science

Introduction

Mini-languages for teaching principles of programming were once among the most exciting innovations for computer science teachers. Mini-languages were introduced at the beginning of 1980 by pioneer works of Richard Pattis in USA (Pattis, 1981), Ivan Tomek in Canada (Tomek, 1982), and similar projects in other countries (Brusilovsky et al., 1997) as an attempt to provide a "gentle introduction" to programming for computer science majors. Just in a few years the approach has won the place in the freshman courses at most respected computer science departments such as Carnegie Mellon, UC Berkeley, and Duke. Yet in another five years we observed a visible decline in the use of Karel-like languages. By mid-1990 it had nearly disappeared from computer science curricula.

Does the disappearance of Karel and similar tools mean that min-languages have no value nowadays and their use is obsolete? The answer is, of course, no. However these changes provide very clear evidence that we need to reconsider the place of mini-languages. The goal of this paper is to discuss the role of mini-languages in the past and new contexts, make some statements about the place of mini-languages now and provide some empirical evidence in support of this statement.

1 Karel the Robot

Karel the Robot, the first and still the most popular mini-language was designed by Richard Pattis as a "gentle introduction" to Pascal for university students taking their introductory programming course. Karel the Robot was completely described in a book with the same name (Pattis, 1981). Pattis also designed the first programming environment for Karel. Karel contains all important Pascal-like control structures and teaches the basic concepts of sequential execution, procedural abstraction, conditional execution, and repetition. The overhead of full high level programming languages is reduced as there are no variables, types, or expressions in Karel. The actor, robot Karel, performs tasks in a world that consists of intersecting streets and avenues, walls, and beepers. Karel can also carry some beepers in his "bag". The main actions of Karel are *move*, *turnleft*, *pickbeeper*, and *putbeeper*. A set of 18 predicates allows Karel to check the state of his world. For example, Karel can determine the presence of nearby walls, if there are any beepers in his bag or at his location, and the direction he is facing. By writing programs that cause Karel to perform carefully selected tasks, students gain experience with the fundamentals while using a pleasant and persuasive metaphor.

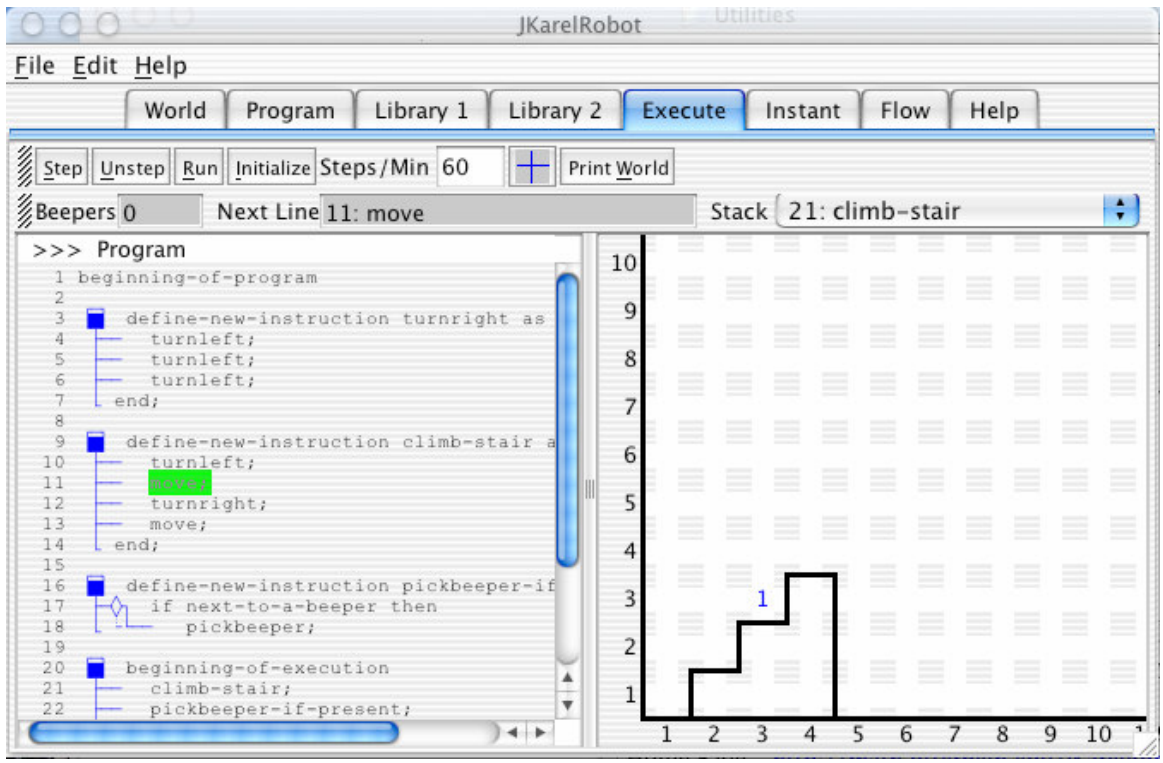


Figure 1. A program for Karel the Robot is executed in a one of the popular environments. Karel's word is shown to the right, the program text is to the left.

The visual power of Karel the Robot language is traditionally supported by special educational programming environments. A typical Karel programming environment is able to execute Karel programs stepwise while showing both the program code and the current state of the world (Figure 1). Each executed command is immediately visualized, i.e., students can see that Karel performs the command (moves, turns, etc.) in the world window and the "current statement marker" in the program window moves to the next command to be executed. A wonderful programming environment Karel Genie developed at the Carnegie Mellon University (Miller et al., 1994) has certainly contributed to Karel popularity in top computer science departments in the USA. Currently, a number of attractive environments for Karel are available. In our study reported below we used a very convenient environment JkarelRobot (Figure 1) that is fully implemented in Java and thus able to run on multiple platforms (Buck & Stucki, 2001).

2 A New Place for Mini-languages

Mini-languages were introduced as a way to provide a "gentle introduction" to programming. Instead of throwing the students into the rich context of real programming languages where core programming ideas and principles were shadowed by troublesome syntax, hidden semantics, and unimportant details, mini-languages offered a simple and clear approach to learn "principles before details" in an attractive visual environment.

These attractive features of mini-languages still provide a great answer for the need to provide a "gentle introduction" to programming. The retreat of mini-languages in the major computer science department was caused by the disappearance of the problem itself.

In the middle of 1980, the author of this paper was successfully using the mini-language Turingal (Brusilovsky, 1991) as an introductory tool in a Pascal-based programming course for computer science freshman at the Moscow State University. The students were enthusiastic and the mini-language was a great help. Yet, in just five years the point of using Turingal was lost. An educational reform promoted by Prof. Ershov and supported by President Gorbatchov has introduced teaching programming in high schools as a part of two-year curriculum in "informatics". Just in two years after the start of this program the situation in freshman computer science classes for majors has changed dramatically. Most of the incoming freshmen came to the university with a reasonable programming background. They had championed programming in high school informatics classes and selected computer science as their major with open eyes. The same situation happened in the USA and a number of European countries

with the introduction of home PCs and advance placement courses. By the middle of 1990 the incoming majors in all strong computer science departments were programming literate. They were able and eager to start with the "real language" and the point of using mini-languages was lost.

Still, we argue that the demand for mini-languages as a way to teach programming principles in the context of introductory programming courses is larger than in their mid 1980 heyday. The dominance of computers in modern life, the force that drew mini-languages from the curriculum of major computer science departments has created a new place for them. Introductory programming classes are now offered to many more students than ten years ago. First, many small colleges started computer science-related degree programs. Second, a number of degree programs from information science and information systems to biology and e-commerce have recognized the need to introduce programming to their majors.

We think that mini-languages are getting a place as tools to introduce the principles of programming for two new cohorts of students – (1) computer science majors in smaller colleges and (2) "non-majors" – students who pursue their majors in a related field. This role is justified by the very same reasons that brought mini-language approach to life in early 1980. The vast majority of these students have no programming experience – exactly as majors of top CS schools in early 1980 – and a mixture of programming principles and details of a specific language appears too thick to swallow in one gulp for most of them.

The Karel place in teaching introductory programming for the first of these cohorts is now commonly appreciated, so our main goal is to promote Karel in non-major courses. The need for a gentle introduction is even greater for non-majors. From one side, the amount of knowledge to learn in an introductory course is increasing with every jump to a new introductory language – from old ALGOL to Pascal to C to C++ to Java. From another side, non-majors are much less eager to work hard on acquiring this knowledge since they do not consider it as one of the necessary assets for their future careers. This paper builds a case for the use of Karel in an introductory programming course for students with non-CS major. It reports our experience in using Karel with Information Science majors, presents the results of a recent formal study, and discusses the outcomes of the study.

3 Karel the Robot for Information Science Majors

The undergraduate Information Science program at the University of Pittsburgh is using Karel the Robot mini-language as part of a new Introductory Programming course IS12. Originally the Information

Science program has relied on introductory programming courses offered by our Computer Science department. However, an analysis of student performance and feedback have shown that these courses oriented to Computer Science majors were not doing a good service for the essentially different cohort of Information Science majors. Our students came unprepared to the demands of more advanced Information Science courses such as Data Structures and File processing. The new course was specifically targeted to provide a "gentle introduction" to programming for this cohort and the use of Karel was a part of this strategy. We allocated about 1/4 of the course for a rather standard pass through Karel using the new edition of the classic book (Pattis, Roberts & Stehlik, 1995) and a popular Java-based Karel programming environment (Buck & Stucki, 2001). The rest of the course was based on C language. Our major goals for using Karel were (1) to provide an introduction to several basic principles of programming and major control structures and (2) to help students with the more complicated C part of the course.

The new course was an instant success. It allowed literally every student who took the course seriously to get a passing grade (C+) and to enter more advanced courses well prepared. Our research concern was, however, to understand to what extent the success of the course can be attributed to using Karel and what was the role of Karel in the course context. We have included a few questions about Karel into a standard course evaluation questionnaire. The two main questions were about Karel help in understanding the programming principles and in getting through the C part of the course. We have also asked the students to leave a free-form feedback about Karel. The results of processing the very first questionnaire were mixed. As we had expected, the students highly evaluated the role of Karel in understanding the principles of programming, however many of them indicated that Karel's help in the C part of the course was low. In addition, we have found that female students were much more positive about Karel than male students. The analysis of the free-form feedback provided by the students provided some hints

To investigate this problem deeper we ran two formal studies in the Fall 2002 and Spring 2003. The next section presents detailed analysis of the studies. The last section of the paper summarizes the whole set of observed results and discusses possible outcomes.

4 The Studies

The goal of our studies was to evaluate the role of Karel as an educational tool in helping the students understand the programming principles and to successfully pass the C part of the course. We were also

attempting to understand how different factors such as past experience, gender, and course grade influenced the student attitude to Karel. We attempted to use both subjective and objective evidence. To evaluate Karel subjectively we administered a specially designed questionnaire. To evaluate it objectively we have attempted to find correlation between student past experience and their performance on Karel part and C part of the course.

The questionnaire administered to the students of our Fall 2002 course included 8 questions: 6 multiple choice questions designed with Likert 5 point scale, one multiple selection question, and one free-form question. The Spring 2003 questionnaire included all these questions plus four additional multiple-choice questions. The questionnaire was administered at the end of the course before the final exam. The participation was voluntary. In total, 29 students out of 42 students of the Fall 2002 group and 35 out of 44 students of the Spring 2003 group chose to participate.

Figure 2 (a and b) summarizes student answers to four critical questions. As we can see, the overall distribution of user answers to these questions is very similar in both semesters despite of some minor variations. This data provides a good ground for our original semi-formal observations about Karel. From one side, students were extremely positive about Karel help in understanding programming principles. About 80% of students provided positive answer to this question; among them about 30% provided highly positive answer. Just 2 students chose negative answers. From another side, they were much less enthusiastic about Karel's help with the C part of the course. Just about 40% of the respondents provided positive answers, a good number provided negative answers, and the remaining third having no strong opinion.

One can argue that the reason for the observable problem is the lack of transfer of knowledge and skills that students have learned in Karel part of the course to the C part of the course. In fact, during the Spring 2003 semesters the teacher put significant efforts into helping with this transfer. When introducing every new C construct (such as while loop) he made clear connection with the similar construct in Karel. It did help improve the student opinion about Karel as a bridge to C learning, but did not change the overall picture. The transfer is important, but we do not think it is the key.

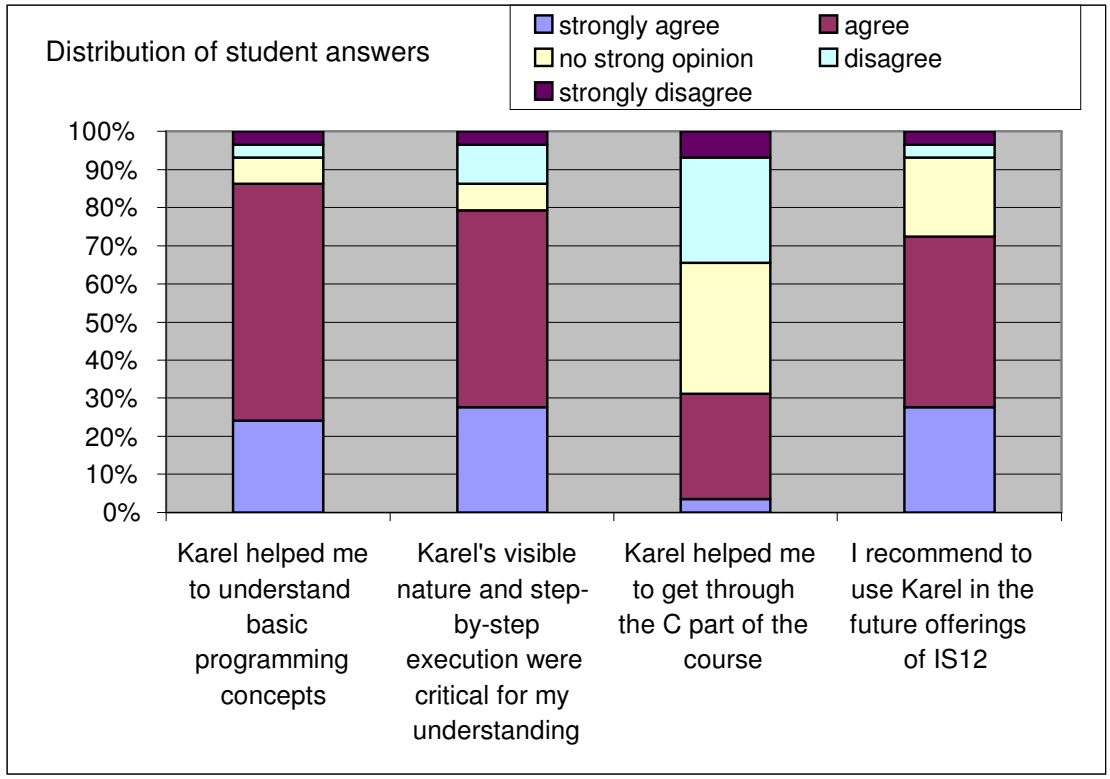


Figure 2a. Distribution of student answers to four critical multiple-choice questions, Fall, 2002.

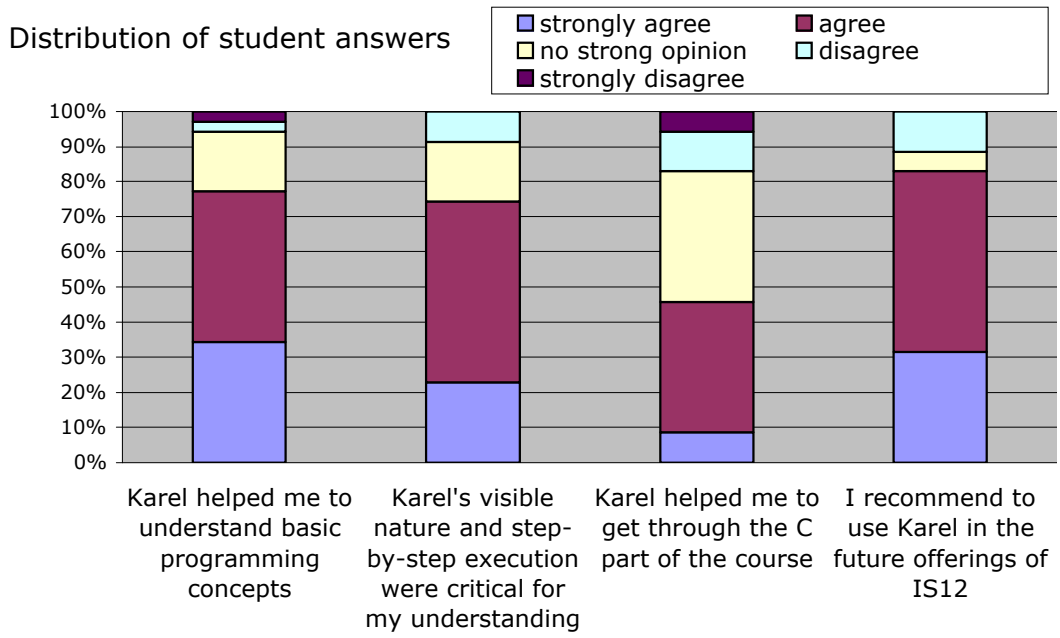


Figure 2b. Distribution of student answers to four critical multiple-choice questions, Spring, 2003.

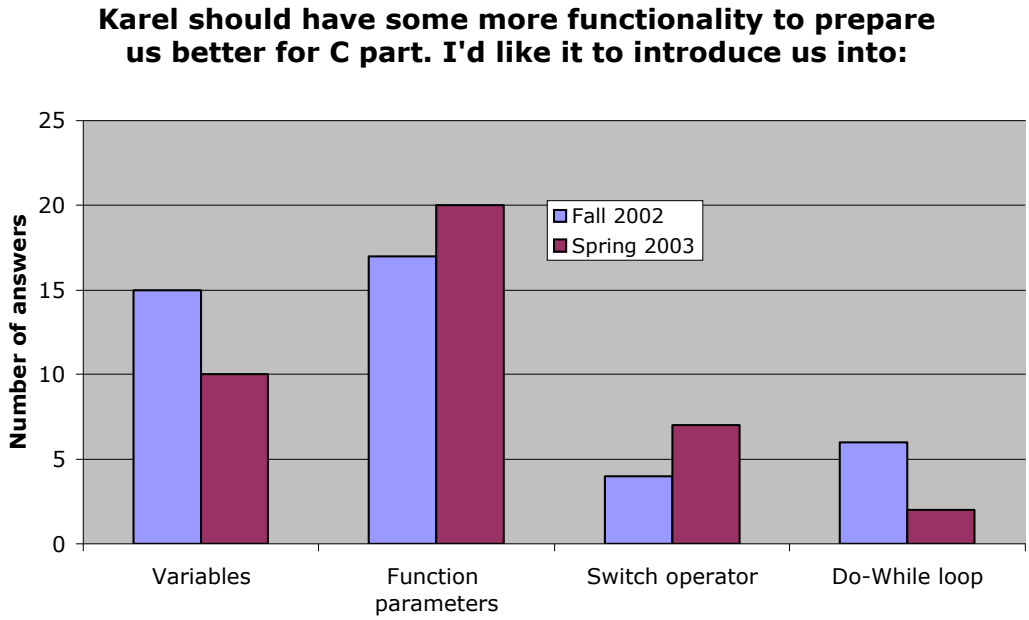


Figure 3. Karel's "missing features"

We think that we have found the real key to the problem while analyzing the free form answers in the earlier questionnaire (Spring 2002). This analysis allowed us to formulate some hypothesis that we attempted to evaluate formally in the reported study. On the positive side, in their free-form answers the students were most often citing the visible nature of Karel and the features of the programming environment that allowed step-by-step execution. Our study shows that the student attitude to Karel's help with the basic principles correlates very well with their attitude to its visible nature. On the negative side, the students most often complained that Karel provided little help in preparation to a number of tough aspects of C language. To evaluate the "incompleteness" of Karel formally we have asked the students to name the concepts and structures where they want Karel to provide more help. The answers have confirmed our observation that the lack of variables and function parameters in Karel was most unfortunate - leaving the students less prepared to face these aspects of C language (Figure 3). Overall C language is known to have a number of hard-to-understand features. Karel helped the students to understand just a few of them – among these Karel's help with while loop was most appreciated. The rest of the features were not introduced in Karel. It's not surprising that in their free-form answers a number of students cited a large difference between Karel and C when arguing that Karel provided little help with C.

To check this observation, we have added three extra questions to our Spring 2003 studies. These three questions attempted to evaluate separately how helpful Karel experience was to understand three specific topics in C programming: variables, functions, and main control structures (if and while). We have expected that Karel can provide good help with if and while (these constructs are almost identical in Karel and C), reasonable help with functions (Karel's commands introduce some aspects of C functions, but not all) and little help with variables and parameters (Karel has none). The results were quite close to our expectations. As Figure 4 shows, students may think that on average, Karel's help with C was much less than its help with programming concepts, however, it helped quite a lot to master some aspects of C that have their close counterpart in Karel. In fact, the students thought that Karel helped them to master *if* and *while* constructs even more than it helped with programming concepts. The help with functions was also regarded higher than the help with C in general. At the same time, the help with variables and parameters was regarded even lower that the help with C in general. This data confirms a rather natural hypothesis: the better a specific programming concept or construct is introduced in Karel, the higher is Karel's help with mastering this concept in C.

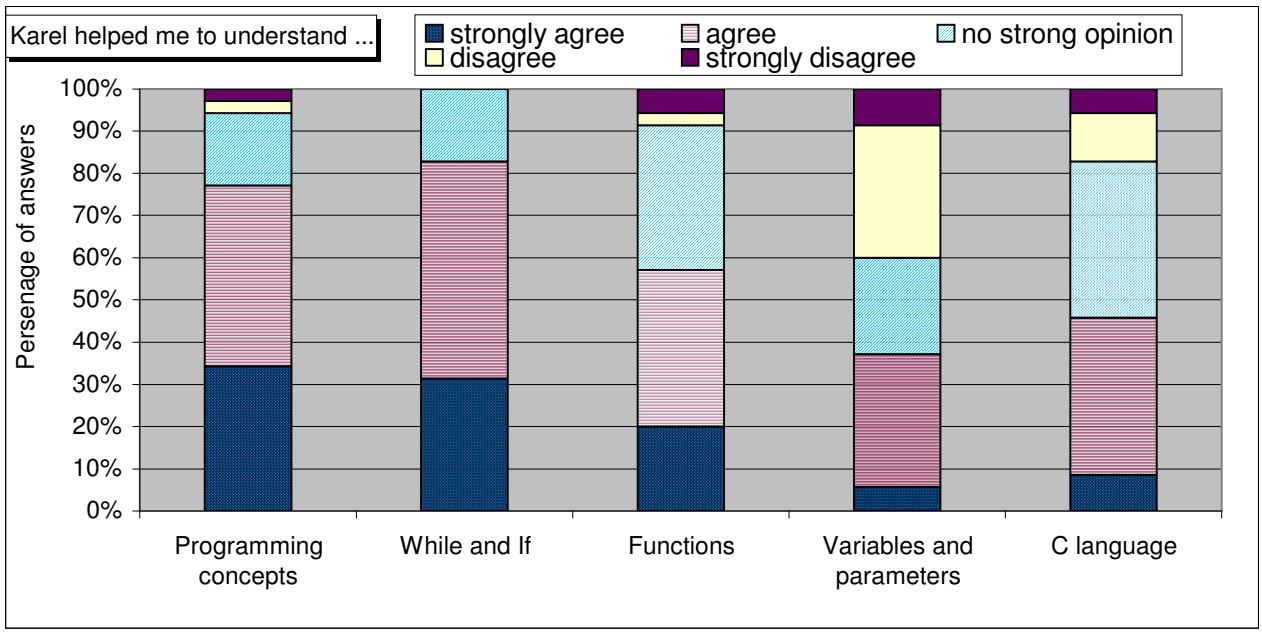


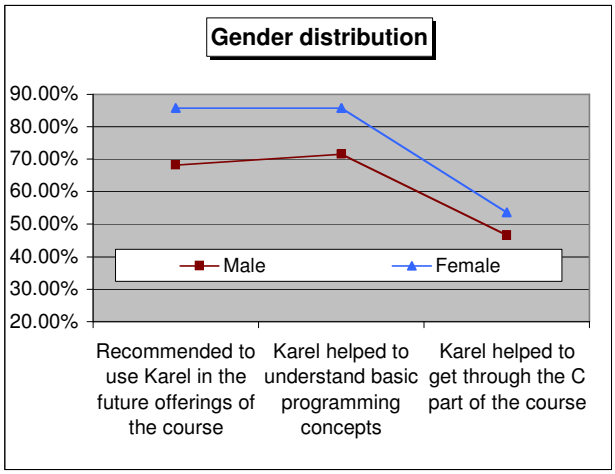
Figure 4. Evaluation of Karel helpfulness for several aspects of the course. Data for Spring 2003.

Yet, despite of the students' mixed attitude to Karel's help with the C part of the course, their overall attitude to Karel in the context of IS12 was as positive as their answer about Karel help with understanding basic concepts. About 80% of students recommended or strongly recommended keeping

Karel as a part of IS12. It looked like Karel's great help with basic concepts overshadowed its less universal value in helping with C language.

One of our early observations that were confirmed by the study was a diversity of student opinions about Karel. Some students appreciated Karel much more than others. Among many answers to the question about Karel help with basic principles there were two negative answers including one strongly negative. At the same time, more than 30% of the answers to the question about Karel help with C were positive, including one strongly positive. To understand why Karel helps some students much more than others, we have attempted to split the student feedback by several criteria such as gender, course performance, and past experience. To make the comparison between the categories easier we have calculated an average "student approval" for each of the evaluated Karel features. A highly positive answer was considered a 100% approval, a positive – 75%, a neutral – 50%, a negative – 25%, and highly negative - 0% approval. Comparing the "approval" of Karel features given by two different groups of students was an easy way to compare the value of Karel for these groups.

Gender: We have observed that female students were visibly more positive about Karel than male students. They have much more enthusiastically recommended using Karel in the context of IS12. Their opinion about Karel help with the basic principles was also almost 10-15% higher than the opinion of male students (Figure 5). For both of these questions all female respondents provided positive or highly positive answers. While the number of participating females was not sufficient to find a significant difference, the striking similarity between Fall 2002 and Spring 2003 data hints that female students, indeed, find Karel more helpful than male students. We may hypothesize that Karel's help is most essential to female students in a programming course and we plan to explore this further.



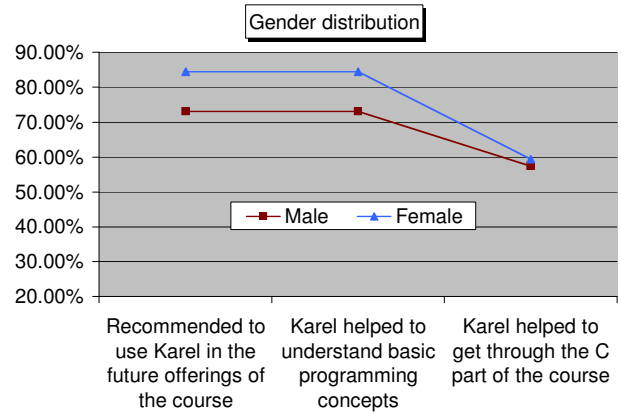


Figure 5. "Approval" of Karel features split by gender. Left: data for Fall 2002, Right, data for Spring 2003.

Programming experience: It was natural to hypothesize that Karel provides the best value for students who had no programming experience. Our questionnaire has collected information about this experience grouping the students into four categories from novices (no programming experience) to those able to write reasonable programs (good programming experience). For both Fall 2002 and Spring 2003 classes, we were not able to find difference between two largest groups with lowest experience (no experience and some experience). It was also hard to make reliable conclusion about the attitude of the users with higher experience since there were very few of them in the introductory programming class. In the Fall 2002 class no users with good experience and only two students with reasonable experience have filled the questionnaire. In the Spring 2003 class, however, 8 students with reasonable experience and 2 students with good experience filled it. As Figure 5 shows, students with reasonable experience rated Karel's help lower that the students with no or some experience in almost all aspects. It seems natural since Karel was targeted at students with little or no programming experience. However, we still can't make a reliable observation that Karel is less appreciated by students with better experience because all 8 students with "some experience" were males. Thus, their lower rating may be influenced by both gender and experience. In contrast, the average rating of the two students with good experience that formed a mixed-gender group (one male and one female) is much closer to the average.

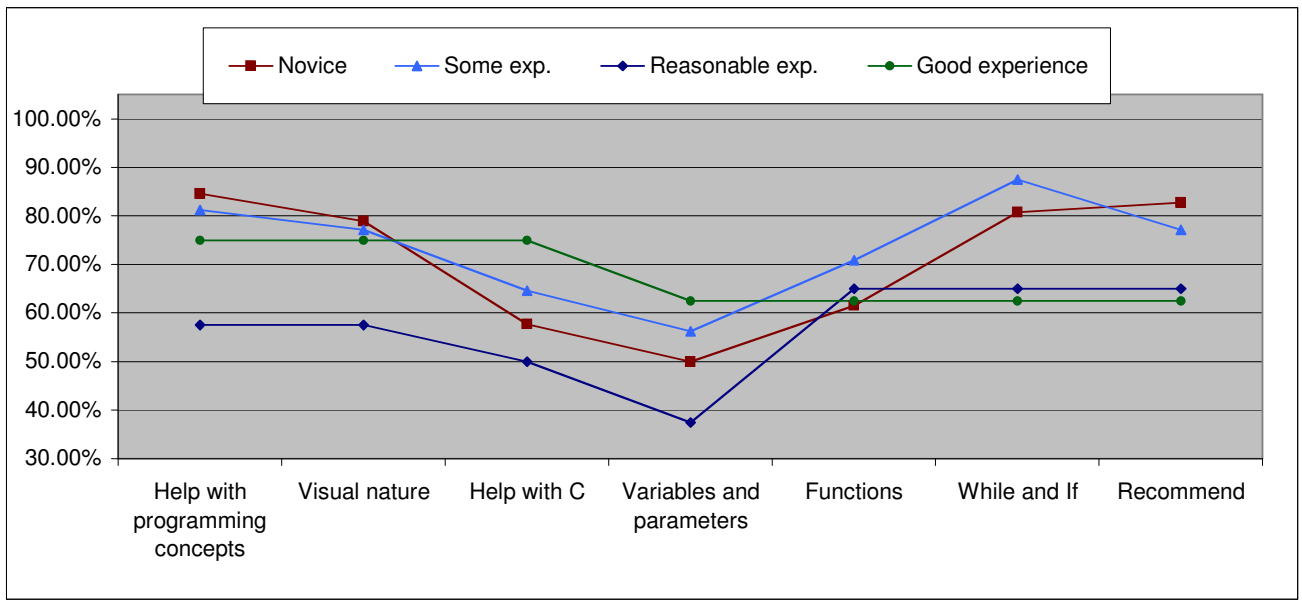


Figure 6. "Approval" of Karel features split by student programming experience. Data for Spring 2003

Student grade: We have also attempted to split the student "approval" by their final course grade (that was not known at the time of filling in the questionnaire). We may speculate that the grade represents "programming abilities" by a student influenced by the amount of efforts devoted to the course. It's natural to hypothesize that students with higher grades will appreciate Karel better. However, our study provided no support for this hypothesis. The profiles of answers given by the groups with A, B, C, or D grades were very close to each other and we can't find any pattern in this data.

Objective evaluation: To find an objective influence of Karel in the context of IS12 course we have attempted to find a correlation between the student past programming experience (as reported in the questionnaire), their performance with Karel, and their performance on the C part of the course. As a measurement of Karel performance we have used the sum of student grades for two Karel-related problems of the class Midterm exam. As a measurement of C performance we have used the total grade for the final exam that was completely based on C language. We hypothesized that both past programming experience and a success on Karel part of the course can contribute to the success on the final exam. To our surprise, a linear regression analysis has found no statistically significant influence of the student past programming experience for both classes. I.e., the amount of knowledge learned during the course has leveled the difference in knowledge about programming that the students had before the course. At the same time, we have discovered a very significant influence of the student Karel performance on their C performance (significance level less than 0.001 for the Fall 2002 and equal to 0.004 for the Spring 2003). I.e., the student performance on the course is determined to a large extent by their performance on the Karel part. We can speculate that the student work with Karel has significantly helped them during C part of the course, though it is not the only way to interpret this data.

5 Discussion

We think that our experience with Karel together with the results of the study provide a good support for the main argument of our paper – the proposal to use Karel in the context of introductory programming courses for non-CS majors. Karel has clearly achieved one of its main goals providing the novices with a gentle introduction into the basic programming principles. To a reasonable extent this success can be attributed to the visible nature and simplicity of Karel.

Karel was also able to help students in getting through the C part of the course, however here the value of Karel was less than we expected. We think that C language has a number of challenges to overcome and Karel has prepared students to meet just a few of these challenges. We think, that providing a more

tight integration between the Karel part and the rest of the course can increase the value of Karel as an introductory language. Alternatively one can consider using a more advanced mini-language in place of Karel, such as, for example, Tortoise language (Brusilovsky et al., 1997) that has simple variables and function parameters that our students found missing in Karel.

Attempting to find a category of students who benefited most from Karel, we have found that female students evaluate Karel and its role in the context of the course much higher than male students do. We can't attribute this fact to female students being just generally "more positive". An evaluation of WebEx (Brusilovsky, 2001), an example-browsing tool performed in the same classroom has shown no differences between opinions of female and male students. Our interpretation is that Karel provides some more critical support for female students taking an introductory programming course. We think that Karel with its clear and visible nature provides an essential help for female students in understanding the principles of programming and basic programming constructs. Informally, we have also observed that with the introduction of Karel into the curriculum, female students became more active participants in class than the male students. For example, in our class where female students constituted about 25%, at least 75% of students who were solving additional programming problems and volunteering to present their solutions to the entire class were females – a rather unusual pattern for a programming class. It gives us a hope that the use of Karel and other visual mini-languages can help female students to overcome the "programming barrier" that they often encounter in a traditional programming classroom with its abstract approach to teaching.

Overall, the data collected so far is very encouraging, allowing us to recommend the use of Karel in any introductory programming classes for non-majors. Ourselves, we plan to continue our work with Karel and other mini-languages trying to find the optimal conditions of using these tools with different categories of students.

References

Brusilovsky, P. (2001) WebEx: Learning from examples in a programming course. In: W. Fowler and J. Hasebrook (eds.) Proceedings of WebNet'2001, World Conference of the WWW and Internet, Orlando, FL, October 23-27, 2001, AACE, pp. 124-129.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., and Miller, P. (1997) Mini-languages: A way to learn programming principles. *Education and Information Technologies* 2 (1), 65-83.

Brusilovsky, P. L. (1991) Turingal - the language for teaching the principles of programming. In: E. Calabrese (ed.) Proceedings of Third European Logo Conference, Parma, 27-30 August, 1991, A.S.I., pp. 423-432.

Buck, D. and Stucki, D. J. (2001) JKarelRobot: A case study in supporting levels of cognitive development in the computer science curriculum. *SIGCSE Bulletin - inroads* (Proceedings of 32nd SIGCSE Technical Symposium on Computer Science Education, February 21-25, 2001) 33 (1), 16-20.

Miller, P., Pane, J., Meter, G., and Vorthmann, S. R. (1994) Evolution of novice programming environments: the structure editors of Carnegie Mellon University. *Interactive Learning Environments* 4 (2), 140-158.

Pattis, R. E. (1981) Karel - the robot, a gentle introduction to the art of programming. London: Wiley.

Pattis, R. E., Roberts, J., and Stehlik, M. (1995) Karel - the robot, a gentle introduction to the art of programming (Second ed.). New York: Wiley.

Tomek, I. (1982) Josef, the robot. *Computers and Education* 6 (3), 287-293.