

# A Proposal for a Worksop for High School Girls

*Evelyn Stiller*

Department of Computer Science  
Plymouth State College  
Plymouth, NH 03264  
estiller@oz.plymouth.edu

June 16, 1999

## **Abstract**

This paper discusses two elements of a workshop for high school girls which will specifically target the difficulty we have in teaching students how to problem-solve and how to relate these skills to actual programming constructs. The first element will address general problem solving skills through the use of reflective cognition in logic game playing. The second element will utilize a software tool to assist students in conceptualizing programming language constructs.

## **Motivation**

The motivation for creating a Summer program for high school girls is the "pipeline" effect that symbolizes the decreasing number of women in computer science starting with the number of women in high school computer classes and ending with the number of full professors in computer science departments [6, 12]. Based on a report from the U.S. Department of Education, National Center for Education Statistics, it becomes apparent that helping undergraduate women succeed during their undergraduate education is critical to the objective of attracting and retaining women in computer science [15]. Based on Fall semester enrollment statistics from this report, the number of women embarking on undergraduate educations have steadily increased from 47.8% in 1976 to 55.7% in 1992. However, only 28.1% of the bachelors degrees in computer science were conferred to women in 1992. Based on the 1996-1997 Taulbee survey [7], the number of women receiving their computer science BS/BA degrees is 16% plus a potential 8.2% of students

with unspecified gender. If approximately half the students where gender was not specified on the survey are assumed to be female, that results in only 20.1% of the bachelor's degrees being conferred to women. Additionally, the percentage of bachelor's degrees in computer science granted to women in 1985 was 37% [2], so the situation has actually eroded over the last thirteen years.

## Workshop

The philosophy behind the workshop is to provide high school girls with a solid foundation for programming and problem solving so that they can thrive in their initial computer science courses. The first element of the workshop focuses on the students' analytical thinking and problem solving skills in a manner that is likely to engage them, namely through game playing. Several games are available for minimal or no cost that exercise students' analytical and deductive thinking skills. An example of using such a game to convey the notion of algorithm development and engage the students in deductive thinking will be presented in detail. Other games and environments are currently being investigated for their inclusion in the workshop as well [5, 9, 11].

The second component of the course addresses students' ability to understand programming language constructs and their ability to apply these constructs effectively during problem solving. Object-oriented programming languages are currently very popular in introductory programming courses but have more overhead than most procedural languages in terms of what is required to write a basic program. Because of the complexity of these languages, students seem to find it difficult to develop a clear conceptual image of the constructs available in these languages. In order to facilitate this end, a tool is proposed here for use during this phase of the class.

The tool to help facilitate strong conceptualization of object-oriented programming language constructs is called the Visual Programming Concepts (VPC) tool. Its goal is to help new, inexperienced, or analytically weak students to have a successful and enjoyable first experience programming, and to reinforce the basic programming language constructs by portraying the results of the programming language instructions visually.

Although a number of initiatives involving program animation [3, 4], the software program introduced here differs from these in some important respects. For example, most program animation initiatives use a compiled language while the VPC tool uses an interpreted language. In addition, the language supported by the VPC is object-oriented while previous initiatives were based on procedural languages such as Pascal or C. Finally, the VPC tool does not require the student to know the precise syntax of the programming language in order to successfully interact with the tool. In fact, the tool itself can be

used to teach programming language syntax. The tool uses formatted windows to help students with syntax and as soon as the student enters an instruction, the VPC tool provides visual feedback concerning the impact of the instruction on computer memory and program flow of control.

By combining the use of the VPC tool and the guided game playing exercises, the workshop proposed in this paper will help to ground students in the skills required for success in computer science.

## Game Playing, Analytical Thinking and Algorithm Development

In our Foundations of Computer Science course, we have tested some of our ideas concerning teaching basic computer science concepts. In particular, this fall, we implemented an exercise to give students and opportunity to develop and utilize their problem-solving skills. In the exercise, students are taught to play the game Sherlock. [14] Sherlock is a logic puzzle computer game with a 6 by 6 array of images. Each block in the array has six possible values and the goal of the game is to use a set of clues to determine the correct position of each of the 36 images. The clues give information like “Image X is in the column next to image Y”, “Image A is in a column somewhere to the left of Image B” and “Image W is in the column between Image J and Image K”.

The kind of logical thinking required for solving a Sherlock puzzle is critical for computer scientists to develop if they are to be successful. Therefore, we believe that by simply playing the game, students are honing skills that will be useful to them in their careers. We can, however, use the game to teach other concepts.

In our conceptualization of the Sherlock assignment, students first learn to play the game. Once they have mastered the game, students must then reflect upon their skill. What kinds of rules do they invoke when faced with a certain type of clue? To solve the puzzle, the student must infer other pieces of information from the clues. For example, if “Image W is in the column between Image J and Image K”, then Image W cannot be in the first column or in the last column of the 6 by 6 array and if “Image A is in a column somewhere to the left of Image B”, then Image A cannot be in the last column of the array and Image B cannot be in the first column of the array. The realization that a clue implies that Image W cannot be in the first column or in the last column of the 6 by 6 array is a rule. We ask students to write down as many of these rules for each type of clue as they can think of during their playing of the game.

Once students have a list of rules, we ask them to reflect upon how they use these

rules to solve the puzzle. In other words, we ask the students to develop an algorithm for how the puzzle is to be solved. Since this course is the first computer science course for many of these students, some of them have difficulty with this step. When they hand in their algorithms, there are some common mistakes, which can be discussed in class to give students a better understanding of what an algorithm entails. For example, many students develop algorithms in which a clue is examined until the positions of all the images involved in the clue have been determined. In reality, one often does not know enough information the first time a clue is encountered to resolve the positions of all the images in the clue. One must move to the second clue before finishing with the first clue. Pointing out such difficulties in class gives students a hands-on example of an algorithm that will not work properly as written. The algorithm is written in English and therefore, students are not intimidated by syntactical concerns. Their focus can instead be on the algorithm itself. In addition, the students can be shown how to walk through their algorithm to determine where there may be difficulties.

Through the use of this fun, game-playing exercise, our students have been able to develop and utilize their problem-solving skills. In addition, we have been able to engage our students in real algorithm design in a context they can readily understand.

## Visual Programming Concepts Tool

The objective of the Visual Programming Concepts (VPC) tool is to create a human-centered software tool to convey programming language concepts. Although learning a traditional, object-oriented language is not deemed to be *human-centered* itself [9, 11], the tool to convey these concepts is intended to be. The term *human-centered* is used here to mean that the tool is structured to accommodate human needs rather than expecting human behavior to conform to the structure of the software. The tool strives to engage the user in an experiential dialog. Such hands-on, interactive dialogs have been shown to be superior teaching scenarios [8, 10, 16]. In Norman's terminology [11], the VPC tool addresses *experiential cognition* because it allows students to experience changes occurring inside the computer in response to their programs. The game-playing exercises using Sherlock, on the other hand, address *reflective cognition* where the students reflect on their experiences to determine how the tasks are accomplished.

The VPC tool helps students conceptualize the basic object-oriented programming language constructs such as variables, memory, pointers, arithmetic expression evaluations, variable scope, method invocation, parameter transmission and flow of control. The VPC tool presents these essential programming constructs in a visual and interactive manner, structured around individual instructions. The components on the screen for each instruction assist the student in building that instruction in a syntactically correct

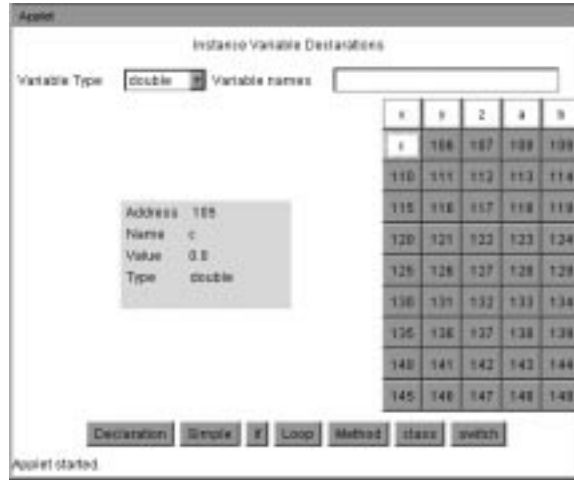


Figure 1: This is the “variable declaration” screen with memory buttons

manner.

For many students, the idea of variables and the manipulation of the values stored in those variables is a somewhat magical concept. Even with current development environments in which memory locations can be queried as to their contents, many students do not truly understand the nature of computer memory and how programming language instructions manipulate and query these locations. Many of our students are visual learners [13]. Because they are unable to actually watch the values in memory locations change in direct response to various instructions, these students find the concept difficult to understand. We believe that being able to actually see these memory locations (or a visual representation of them) and how the values in them change as instructions are executed will help many students truly understand this basic programming concept.

The VPC tool is not a visual programming language in the conventional sense [9], where a programmer manipulates icons in order to create a program. The VPC tool is a graphical environment in which visual feedback conveys the result of a program instruction. The VPC tool contains a language interpreter for a subset of the Java programming language [1]. The tool presents a visual representation of memory as a table of buttons, where each button represents a memory cell. For each Java statement entered by the student programmer, the appropriate change occurs immediately on the visual representation of computer memory. For example, if the user declares a floating-point variable named *Y*, the tool will show that one of those buttons now has the name *Y*, and its background color will change to draw the user’s attention to the fact that the location is now allocated to the current program. The resulting screen format is shown in Figure 1.

The VPC tool dedicates a single screen to each category of programming statements.

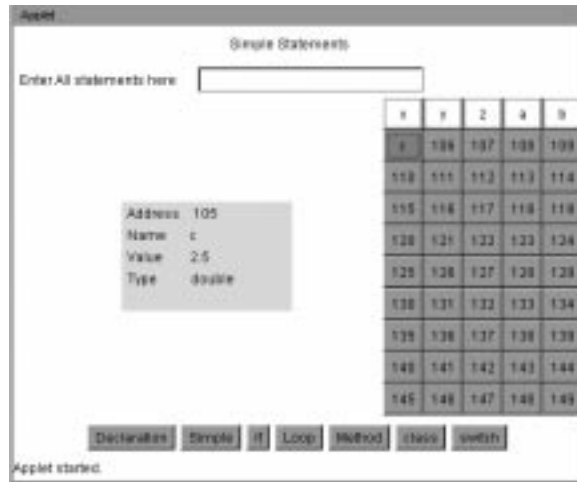


Figure 2: This is the “assignment” screen with memory buttons

Therefore, each screen is structured to help the student with the syntax of the particular statement. As shown in Figures 1, 2 and 3, each screen has a series of buttons at the bottom that allows the student to begin entering a particular type of Java instruction. The main screen, shown in Figure 1, allows the entry of a variable declaration and thus, contains a drop box of all the primitive types. This list informs the student of the primitive types supported by Java and prevents errors due to typing mistakes. Figure 2 shows the screen that allows simple statements such as assignment statements, and Figure 3 illustrates the screen that accepts an if statement.

In addition to the primary window which shows the screen structured around a single program instruction, when the VPC tool is running a secondary window is displayed which shows the aggregate program being formulated by the user. This secondary window presents the entire program in order to allow the student to reposition herself during the programming process, and to portray program flow of control during program animation. This program animation visually portrays the essential concept of program flow of control.

Another essential concept that students typically have difficulty with is the idea of variable scope. In the secondary window showing the aggregate program, all variables in scope at the current line of code (determined by the current position of the cursor) are displayed. If the student programmer clicks on the aggregate program window to change the placement of the cursor, the corresponding scope change will be reflected in the variables shown in the table of memory buttons. Thus, students can experiment with their understanding of scope by clicking in various places in the program and viewing the corresponding memory definition changes.

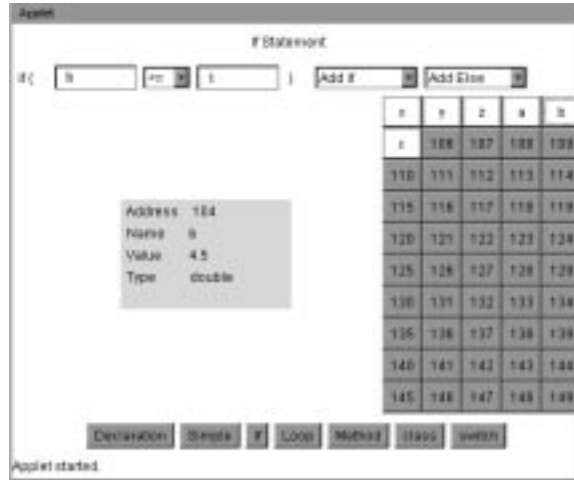


Figure 3: This is the “if” screen with memory buttons

## Conclusion

In this paper, we have presented two elements of a workshop for high school girls interested in math and computer science. These components attempt to address some of the difficulties students have encountered in later computer science courses when students have not been properly grounded in problem solving, algorithm development and basic programming constructs. Through a game-playing exercise, students practice logically determining the solution to a puzzle. When they have become sufficiently skillful at solving the puzzle, they reflect on their experiences and develop an algorithm for solving the puzzle. In the second component, students learn a subset of Java through the use of a software development tool that provides immediate feedback concerning the impact of various programming constructs on memory locations. The VPC tool currently exists as a prototype and is being programmed as a robust, reliable software product in software engineering classes over the next year.

## References

- [1] Arnold, Ken and Gosling, James. *The Java Programming Language, Second Edition*. Addison-Wesley, Reading, MA, 1997.
- [2] Association, Computing Research. *Statistical Trends in Computer Science*. <http://www.cra.org/statistics/trends/>, (World Wide Web Page), 1998.
- [3] Biermann, A. W., Fahmy, A. F., Guinn, C., Pennock, D., Ramm, D., and Wu, P. Teaching a hierarchical model of computation with animation software in the

- first course. In *Proceedings of the Twenty-Fifth SIGCSE Technical Symposium on Computer Science Education*, Phoenix, AZ, March 6-12 1994.
- [4] Boroni, C. M., Eneboe, T. J., Goosey, F. W., Ross, J. A., and Ross, R. J. Dancing with dynalab endearing the science of computing to students. In *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, pages 135–139, Philadelphia, PA, February 15-18 1996.
- [5] Bruckman, A. and Resnick, M. The mediamoo project: Constructionism and professional community. *Convergence*, 1(1):94–109, 1995.
- [6] Frenkle, K. Women and computing. *Communications of the ACM*, 33:34–46, 1990.
- [7] Kozen, Dexter and Zweben, Stu. *Computing Research Association: Statistical Trends in Computer Science*. [http://www.cra.org/statistics/survey/97/1996-1997\\_taulbee\\_survey.html/](http://www.cra.org/statistics/survey/97/1996-1997_taulbee_survey.html/), (World Wide Web Page), 1998.
- [8] Laurel, B. *Computer as Theater: A Dramatic theory of interactive experience*. Addison-Wesley, Reading, MA., 1991.
- [9] Nardi, B. A. *A Small Matter of Programming*. MIT press, Cambridge, MA, 1993.
- [10] Norman, D. A. Cognitive artifacts. In Carroll, J. M., editor, *Designing Interaction: Psychology at the Human-Computer Interface*, pages 17–382. Cambridge University Press, New York, 1991.
- [11] Norman, D. A. *Things that Make Us Smart*. Addison-Wesley, Reading, MA, 1993.
- [12] Pearl, A., Pollack, M. E., Riskin, E., Thomas, B., Wolf, E., and Wu, A. Becoming a computer scientist. *Communications of the ACM*, 33:47–57, 1990.
- [13] Reiff, Judith C. *Learning Styles*. National Education Association, Washington DC, 1992.
- [14] Software, Everett Kaser. *Sherlock for Windows*. <http://www.teleport.com/~everett/sherwin.html>, (World Wide Web Page), 1998.
- [15] U.S. Department of Education, National Center for Education Statistics. *Women: Education and Outcomes*. NCES 96-061, by Chan Kopka, Teressita and Korb, Roslyn, Washington DC, 1996.
- [16] Zhang, J. The interaction of internal and external representations in a problem solving task. In *Proceedings of the 13th Annual Conference of the Cognitive Sciences Society*, pages 954–958, Chicago, 1991.