USING JEROO TO INTRODUCE OBJECT-ORIENTED PROGRAMMING

Brian Dorn[1] and Dean Sanders[2]

Abstract *– The authors present Jeroo, a pedagogical tool that provides a gentler introduction to object-oriented programming. Jeroo has been developed to help novice programmers learn the basic notions of using objects to solve a problem, learn to write methods that support a functional decomposition of the task, and learn the semantics of fundamental control structures. Jeroo's syntax provides a smooth transition to Java, C++, or C#. The user interface has a single window in which everything is always visible. Source code highlighting, simple animation, and a continuously updated status panel provide a rich teaching and learning environment. Jeroo has been class tested at Northwest Missouri State University, and has proven to be an effective tool for working with novice programmers. Used in the first four weeks of a Java programming class, Jeroo produced a significant increase in student comfort and confidence levels, especially among female students. Jeroo and user documentation are available at http://www.nwmissouri.edu/~sanders/Jeroo/Jeroo.html.*

Index Terms – computer science education, introductory programming course, object-oriented programming, pedagogy.

INTRODUCTION

Computer programming has always been difficult to teach. Whether we are teaching procedural or object-oriented programming, the first few weeks can be intimidating as the students work to master abstract concepts and a myriad of details. Many students lose interest and self-confidence in the early weeks of their first programming course. Students tend to lose interest, partly because they grew up with video games and graphical user interfaces, while typical programming activities involve text-based I/O, perhaps disguised with simple message and dialog boxes. They tend to lose self-confidence, partly because many of the concepts are abstract and difficult to visualize or understand. This paper introduces Jeroo, a tool that helps novice programmers maintain their interest and self-confidence while mastering important concepts.

Jeroo is an integrated development environment and microworld that was inspired by Karel the Robot[8] and its descendants. Jeroo was designed to help students learn to instantiate and use objects, learn to design and write methods, and learn about control structures. Students who work with Jeroo report increased confidence in their ability to learn computer programming. Faculty members who have used Jeroo in their courses report that it engages the students immediately and that students appear to master important concepts faster than they did when Jeroo was not used. Jeroo is suitable for use in an introductory programming course, or in a computer literacy course.

EDUCATIONAL GOALS FOR JEROO

Four educational goals provided an overall context for developing Jeroo:

- Engage students immediately
- Generate confidence
- Improve comprehension of selected topics
- Develop crucial programming skills

Three aspects of Jeroo help to engage the students early: the metaphor, the design of the user interface, and the animated execution. These aspects are described in subsequent sections of this paper.

Three other aspects help the students gain confidence in their ability to learn to write programs: the anthropomorphic nature of the Jeroos, the feedback provided by animated execution with simultaneous highlighting, and the ability to execute a program stepwise or continuously at varying speeds. These aspects, also, are described in subsequent sections.

Keeping in mind that Jeroo was to be used for just three or four weeks in a course, we restricted the Jeroo programming language to emphasize a few significant features. The specific language features are described later in this paper.

Finally, we tried to design Jeroo to be a tool that could help the teacher demonstrate and the students learn crucial programming skills such as problem analysis and decomposition, incremental development of algorithms, and formulation of pre- and post-conditions. Our experience in using Jeroo indicates that we have met this design goal.

THE METAPHOR

A Jeroo is a rare kangaroo-like animal living on Santong Island, a remote speck of land in the South Pacific. Jeroos have an unusual way of moving about their island; they move by hopping, but they can only hop in the four main compass directions. The island is also home to the large Winsum flowers that constitute the primary food source for

[1] Brian Dorn, Iowa State University, Ames, IA 50011 dorn@cs.iastate.edu
[2] Dean Sanders, Northwest Missouri State University, Maryville, MO 64468 sanders@mail.nwmissouri.edu

0-7803-7961-6/03/$17.00 © 2003 IEEE

the Jeroos. A Jeroo can pick, carry, and plant the flowers. The Jeroos must be on the lookout for nets that have been set by a trapper who is seeking new specimens for a zoo.

All students understand this metaphor, immediately. They feel comfortable with the metaphor because it is nontechnical, is nonviolent, and has an anthropomorphic familiarity.

THE USER INTERFACE

The user interface consists of a single window in which all components are visible at all times, making it easier to associate a program with its execution. The use of menus and a tool bar help to provide a familiar interface that the students can learn to use with virtually no instruction. Figure 1 shows the appearance of the user interface for a running program.

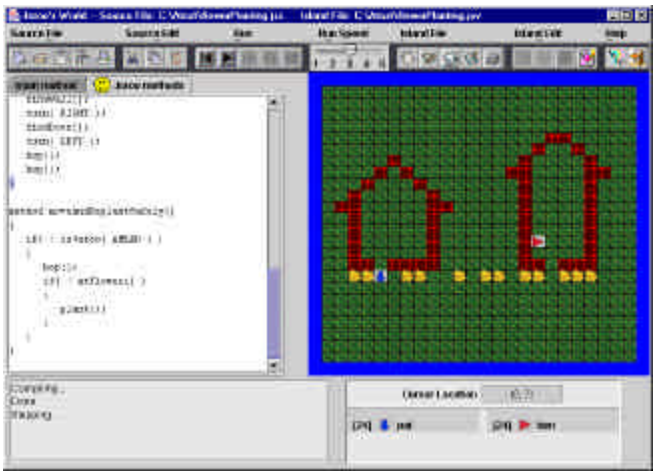

FIGURE 1
THE USER INTERFACE FOR JEROO

The left-hand portion of the screen contains tabbed panes for editing the source code. One pane is used for a main method where Jeroos are instantiated and used to solve a specific problem. The other pane is used for user-defined Jeroo methods that extend the behavior of all Jeroos. The source code editor supports common editing features: cut, copy, paste, undo, and redo.

The right-hand portion of the screen contains a representation of Santong Island. Students use a point-and-click process to place flowers and nets on the island prior to running a program. A cursor location panel helps the students locate specific cells on the island. The island area also displays a simple animation of a running program.

The user interface includes two status panels. A message area is used to display status messages, syntax error messages, and runtime error messages. A Jeroo status panel is continually updated to display the state of every Jeroo in a running program.

THE JEROO LANGUAGE

The Jeroo language was designed so that its syntax would mirror the common syntax of Java, C++, and C#. This syntax allows a teacher to say "The Jeroo language is really a subset of Java (or C++ or C#)", thereby avoiding the question "Why we are studying Jeroo instead of a 'real' programming language?" The Jeroo language was designed to focus on three major items: objects, methods, and control structures.

The Jeroo class is the only one that is available within the language. There are no data types and no variables other than references to Jeroo objects. There are, however, eight predefined directional constants. Relative directions are specified by the constants LEFT, RIGHT, AHEAD, and HERE. Compass directions are specified by NORTH, EAST, SOUTH, and WEST. Each Jeroo object has three attributes: location, direction, and number of flowers. These attributes are always visible, and can be affected by constructors and some of the predefined methods.

A student may instantiate up to four Jeroos in a program. The Jeroo class has six constructors that allow a student to specify the values of various attributes. These constructors provide a gentle introduction to the use of arguments and the concept of overloading.

The Jeroo class has several predefined methods that can be partitioned into two categories: sensor methods and action methods. These allow a Jeroo object to examine and interact with its immediate surroundings.

The sensor methods are essentially boolean methods that provide information about a Jeroo's immediate surroundings. For example, they provide answers to questions like: "Is there a net to the right of this Jeroo?" One of these methods, *hasFlower()*, takes no arguments. Four of them, *isFlower.(…)*, *isNet(…)*, *isWater(…)*, and *isJeroo(…)*, require that a relative direction be provided as an argument. Lastly, the method *isFacing(…)* requires a compass direction as an argument. These six sensor methods are used to construct conditions for *if* and *while* statements.

The Jeroo language supports three fundamental control structures: *while*, *if*, and *if-else*. The condition for each of these statements is always constructed from one or more sensor methods and the operators &&, ||, and !.

Action methods allow a Jeroo to move about the island and interact with any flowers and nets that it may encounter. The *hop()* and *turn(relative_direction)* methods allow a Jeroo object to move about the island. A Jeroo may alter its environment by picking up flowers, planting flowers, and disabling nets with the *pick()*, *plant()*, and *toss()* methods, respectively.

A programmer may define additional action methods to extend the behavior of the Jeroos. The user-defined Jeroo methods lack arguments, but they can call other methods. They can also be recursive, but we do not introduce that feature in our courses. The ability to write additional

methods is an effective way to introduce modularity at an early stage.

THE RUNTIME BEHAVIOR

The runtime module, which is invoked from a menu selection or toolbar button, provides both language translation and visual execution of a Jeroo program. This module illustrates the connection between source code and the Jeroo's behavior.

A program may be executed stepwise or continuously at one of five speeds. In either mode, the source code is highlighted as the program is being run, and the actions of the Jeroos are animated simultaneously. The students may switch modes and speeds at will during the execution of a Jeroo program.

The runtime module also updates two status panels. One panel displays textual information about the status of the program (Compiling, Running, etc), while the other displays the identifier, direction, and number of flowers associated with each Jeroo object.

The combination of source code highlighting, animation, and status information creates a rich learning environment. Syntax and runtime errors can be located quickly; the semantics of the control structures are readily apparent; and the interaction between methods is revealed.

PERCEIVED BENEFITS

We have observed several benefits related to using Jeroo after using it as part of the curriculum for two consecutive semesters. Benefits can broadly be categorized as programming concepts, programming practices, and student satisfaction. While evaluation of the first two categories remains anecdotal, student surveys have allowed us to objectively assess satisfaction.

In terms of programming concepts, we feel that students have a better grasp of control structures, methods, and objects following the four-week introduction to programming using Jeroo. Our former approach to teaching this course did not allow for the earlier and more frequent use of these concepts that Jeroo provides. The enhanced understanding of these most basic concepts has allowed students to program with the same confidence when the course material moves into Java during the fifth week.

We have also noticed that students have developed a more mature style of programming than in the past. By the end of the semester, most students have begun to decompose problems and plan before they touch the keyboard. We attribute this to their previous interactions with Jeroo, in which the planning and decomposition tasks are more natural than in a text and number processing, command-line environment. Students also appear to be better at reading and tracing programs. We believe this is due to the fact that tracing was emphasized with Jeroo, where programs tend to be easier to read. Additionally, the source code highlighting during execution allows students to see a real connection between source code and changes in program state.

Increases in student confidence, comfort, and satisfaction are perhaps the most important benefits we have observed. Not only do increases in these factors play an important role in the number of students who continue on to future programming courses, but they also reinforce the fact that Computer Science is not a field in which only a relatively small, culturally monotone, and stereotypical group of individuals can succeed. In other words, it's not just for nerds.

Student satisfaction was assessed via a questionnaire given shortly after the transition from Jeroo to Java had been made. This survey solicited demographic information and asked students to respond to the following questions using five-point Lickert scales (1=low, 5=high).

- Rate your confidence in your ability to learn computer programming before you started the course.
- Rate your confidence in your ability to learn computer programming after you finished the Jeroo portion of the course.
- Rate your level of comfort about this course before the semester began.
- Rate your level of comfort about this course after working with Jeroo.
- Do you think that working with Jeroo was worthwhile to you?
- Do you think that working with Jeroo was worthwhile to your classmates?
- Do you think that working with Jeroo was a good way to lead into Java?

In total, 97 students completed this survey over the course of two semesters. Mean values for several of these questions are presented in Table I.

TABLE I
SUMMARY OF QUESTIONNAIRE RESULTS

Type	Increase in Confidence	Increase in Comfort	Worthwhile to You	Worthwhile to Others	Good Intro to Java
Overall	0.52	0.60	3.62	3.89	4.04
		Gender			
Female	0.95	1.18	4.16	3.95	4.26
Male	0.40	0.42	3.45	3.87	3.96
		Programming Experience			
None	0.79	0.69	3.74	3.70	4.13
Some	0.48	0.75	3.72	3.99	3.88
Full Course	0.26	0.42	3.41	3.88	3.94
Self-Taught	0.35	0.59	3.53	4.30	4.23

As illustrated in the data shown above, all groups of students, regardless of gender, or programming experience, found Jeroo to be a valuable tool for themselves and others, and felt it served as a good introduction to Java. Students

with more programming experience found less value in Jeroo, as expected. Interestingly, however, students with more programming experience felt that Jeroo was more valuable to others. This would suggest that students who are more familiar with programming feel that it effectively teaches the fundamental concepts.

One unexpected result was that males and females had seemingly different views about Jeroo. Notice from the data that the variation between increases in confidence and comfort are quite large. Additionally, females found the tool more valuable in general than the males did. Figures 2 and 3 explore the gender differences in more detail.

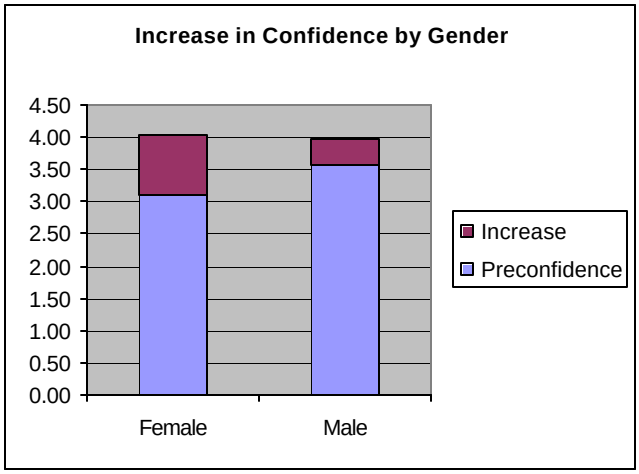

FIGURE 2
INCREASE IN CONFIDENCE BY GENDER

The stacked bar graph depicts the dramatic effect using Jeroo had on female confidence levels. While females started with less confidence than the males, their average confidence levels were nearly identical following the use of Jeroo. A similar effect is presented in figure 3 with respect to student comfort levels.

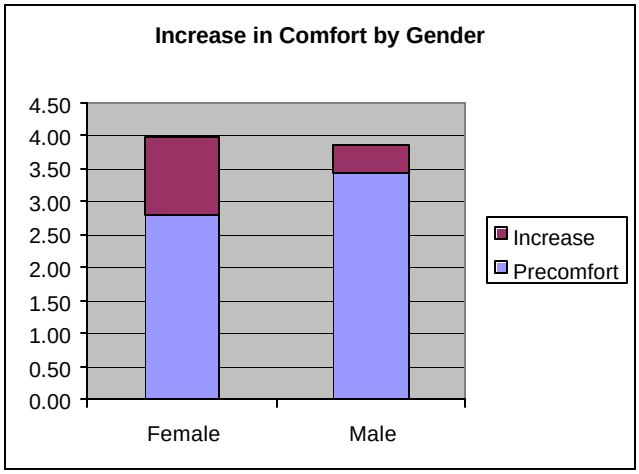

FIGURE 3
INCREASE IN COMFORT BY GENDER

Initially female confidence and comfort levels were much lower than males. After using Jeroo, these levels were raised to approximately equal to or slightly greater than that of the male students. The potential ramifications of this simple observation are very important. A recent study [4] indicates that females have lower confidence in their computing abilities than men, even when statistically controlling for quantitative ability. Additionally, this decreased confidence may lead to a decrease in the likelihood that females choose Computer Science as a major and an increase in the likelihood that women will drop out of CS programs. If Jeroo can level the playing field of confidence early on, it may lead directly to an increase in the number of women who choose CS as their field of study.

Using a paired t-test to compare pre and post confidence levels overall, the probability there was no significant increase in values is less than 2.0×10^{-8}. We conclude that the mean confidence level did increase after using Jeroo. Similarly the p-value associated with a comparison of the comfort levels is approximately 1.4×10^{-8}. Thus, the average comfort level was greater after learning with the Jeroo tool. These results indicate that Jeroo is not only a valuable tool in increasing confidence and comfort of a minority audience in CS, but also that it provides similar increases in these levels for all students.

Beyond the numbers, Jeroo has served as a means to excite students about computing. Students have become more enthusiastic about the coursework; they see programming as fun. This can be attributed to the game-like nature of Jeroo. Its graphically rich environment presents users with a far more exciting platform to learn about basic concepts than previous teaching methods. However, this can also make the transition to a full language complicated. In order to maintain the level of excitement, careful planning must be made to ensure that future assignments are equally gratifying. We feel that the extra work involved in this stage is well worth the payoff in student satisfaction.

COMPARISON TO RELATED TOOLS

The original Karel has sired several descendants that support an object-oriented style of programming. Those who have used these tools report results similar to those we have observed. This section compares Jeroo to a few representative tools. It is not intended to be a complete list.

Karel++ [2] is an object-oriented descendant of the original Karel. Karel++ was designed as the entry point for students of C++. Karel J. Robot [3] has been developed recently to be a Java-based sibling of Karel++. JKarelRobot [5, 6] is yet another descendent of Karel. This tool supports three different programming languages: C++, Java, and LISP. The Java based version is also known as Jarel. Finally, students at the University of Waterloo use another descendant of Karel [1]. This version was created as a Java package that can be imported into a standard Java program.

The Jeroo tool has a syntax that mirrors the intersection of Java, C++, and C#. Karel and its descendants have a syntax that is rooted in Pascal.

The Jeroo tool uses a single window in which all components are visible at all times. Integrated development environments exist for Karel++, Karel J. Robot, and JKarelRobot, but these tend to use multiple windows or screens.

Jeroo's island is modeled after a two-dimensional array. The island is bounded on all sides. A location on the island is specified as a (row,column) pair. The row and column numbering begins at zero in the northwest corner. This provides a nice introduction to zero-based counting, provides a preview of the way we typically visualize a two-dimensional array, and corresponds to a typical screen coordinate system. Karel's world is modeled after the first quadrant in a Cartesian coordinate plane. Karel's world extends infinitely far to the North and East. A location in Karel's world is specified as a (row,column) pair. The row and column numbering begins at one in the southwest corner.

A condition in Jeroo's language is formed by invoking sensor methods and combining them with logical operators. A condition (test) in Karel's language is just one of several keywords. In general, Karel has no logical operators.

SUMMARY

Jeroo has proven to be an effective tool for teaching the concepts of objects, methods, and control structures to novice programmers. The single-screen development environment is easily mastered. The animated execution and concurrent code highlighting aid understanding and help maintain interest. The carefully chosen syntax provides a smooth transition into Java, C++, or C#. The use of Jeroo allows us to cover the same topics; we just teach them differently and in a different order. If the use of Jeroo is carefully planned within the context of a total course, the result is an improved learning experience.

REFERENCES

[1] Becker, B. W. Teaching CS1 with Karel the Robot in Java. *Proceedings of the Thirty-Second SIGCSE Technical Symposium*, Vol. 32, No. 1., February, 2001, pp. 50-54.

[2] Bergin J., Stehlik, M., Roberts, J. and Pattis, R. *Karel++: A gentle Introduction to the Art of Object-Oriented Programming*, John Wiley & Sons, 1997.

[3] Bergin J., Stehlik, M., Roberts, J. and Pattis, R. *Karel J. Robot: A gentle Introduction to the Art of Object-Oriented Programming in Java*, 2002. Online. Internet. Sept. 6, 2002. Available WWW: http://csis.pace.edu/%7Ebergin/KarelJava2ed/Karel++JavaEdition.html

[4] Beyer, B., Rynes, K., Perrault, J., Hay, K., and Haller, S. Gender Differences in Computer Science Students. *Proceedings of the Thirty-Fourth SIGCSE Techincal Symposium*, Vol 35, No. 1, February 2003, pp. 49-53.

[5] Buck, D. and Stucki, D. JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. *Proceedings of the Thirty-Second SIGCSE Technical Symposium*, Vol. 32, No. 1., February 2001, pp. 16-20.

[6] *Karel the Robot*, 2002. Online. Intemet. Sept. 6, 2002. Available WWW: http://math.otterbein.edu/JKarelRobot

[7] Kölling, M. and Rosenberg, J. An Object-Oriented Program Development Environment for the First Programming Course. *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium*, Vol. 27, No. 1., February 1996, pp. 83-87.

[8] Pattis, R. E. *Karel the Robot: A gentle Introduction to the Art of Programming*, 2nd ed., John Wiley & Sons, 1995.

[9] Sanders, D. and Dorn, B. Classroom Experience With Jeroo. *Journal of Computing Sciences in Colleges*, Vol. 18, No 4., April 2003, pp. 320-328.

[10] Sanders, D. and Dorn, B. Jeroo: A Tool for Teaching Object-Oriented Programming. *Proceedings of the Thirty-Fourth SIGCSE Technical Symposium*, Vol. 35, No. 1., February 2003, pp. 201-204.

APPENDIX A: A SAMPLE PROGRAM

A Jeroo starts in the southwest corner facing East. The Jeroo must jump over an unspecified number of hurdles with random heights and spacing to locate and pick a flower. A possible layout for the island is shown in Figure 3.

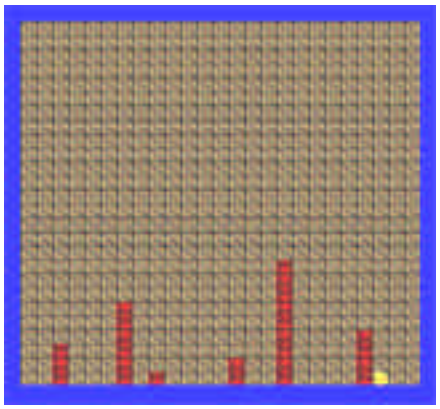

FIGURE 3
POSSIBLE ISLAND LAYOUT

```
method main() {
  Jeroo kim = new Jeroo(23,0);
  while (!kim.isFlower(HERE)) {
     kim.run();
     if( kim.isNet(AHEAD) ) {
        kim.jumpHurdle();
     }
  }
  kim.pick();
}

======= user-defined methods =======

method run () {
  while( !isNet(AHEAD)  &&
         !isFlower(HERE)  )
    hop();
}

method rise() {
  while( isNet(RIGHT) )
    hop();
}

method descend() {
```

```
  while( !isWater(AHEAD) )
    hop();
  turn(LEFT);
}

method jumpHurdle() {
  turn(LEFT);
  rise();
  turn(RIGHT);
  hop();
  hop();
  turn(RIGHT);
  descend();
}
```