CLASSROOM EXPERIENCE WITH JEROO*

Dean Sanders
Computer Science/Information Systems
Northwest Missouri State University
Maryville, MO 64468
sanders@mail.nwmissouri.edu

Brian Dorn
Department of Computer Science
Iowa State University
Ames, IA 50011
dorn@cs.iastate.edu

ABSTRACT

Jeroo is a tool that has been developed to help students in beginning programming courses learn the semantics of fundamental control structures, learn the basic notions of using objects to solve problems, and learn to write methods that define the behavior of objects. Jeroo is similar to Karel the Robot and its descendants, but has a narrower scope and has a syntax that provides a smoother transition to either Java or C++. Jeroo has been class tested at Northwest Missouri State University, and has proven to be an effective tool for working with students in a beginning programming class.

1 INTRODUCTION

The first programming course has always been difficult. Whether we are teaching procedural or object-oriented programming, the first few weeks can be intimidating to novice programmers as they work to master abstract concepts and a myriad of details. Many students lose their self-confidence in these early weeks of a beginning programming course. This paper introduces Jeroo, a tool that helps novice students maintain their confidence while mastering important concepts early in a course.

Jeroo is an integrated development environment and microworld that was inspired by Karel the Robot[6] and its descendants. Jeroo was designed to help students learn to instantiate and use objects, to learn to design and write methods, and to learn about control

---

structures. Students who work with Jeroo report increased confidence in their ability to learn computer programming.

2 ORIGINS OF JEROO

The origins of Jeroo can be traced to the mid 1980's, when the first author of this paper introduced Karel to a beginning programming class at Illinois State University. Lacking appropriate hardware to have the students work with a simulator, all programs were handwritten. The lack of a simulator made it easy to implement the students' suggestions for modifications to both the metaphor and the language. The metaphor evolved to kangaroo-like animals hopping around an island picking flowers and avoiding nets. The language was modified to fit the metaphor and to add logical operators.

In 1990, Lai Kuan Tong completed work on Jessica, which she developed as her master's project at Illinois State University. Jessica is a development environment and simulator for the metaphor that was described above. The results of her work were never published, but her Jessica tool was used successfully for several years at Illinois State University.

In 1999 and again in 2001, the first author of this paper worked with students in software engineering classes at Northwest Missouri State University to define the requirements and construct a prototype for an object-oriented successor to Jessica. The authors of this paper used those requirements and that prototype as the basis for constructing Jeroo. The following design goals guided our efforts and helped curb our enthusiasm for adding more features.

- Design the tool for novice programmers.
- Keep everything as straightforward as possible.
- Limit the tool's scope to just a portion of the course.
- Focus on just control structures, methods, and objects.
- Eliminate variables and data types.
- Keep the Jeroo syntax close to Java and C++.
- Provide animated execution and code highlighting.
- Keep everything visible at all times.
- Allow sophisticated problems to be solved without complicated code.

3 CAPABILITIES OF JEROO

The Jeroo tool has four major components: (1) the user interface, (2) the Jeroo programming language, (3) editors, and (4) a runtime module. These components are described in the following subsections.

3.1 The User Interface

The user interface consists of a single window in which all components are visible at all times (see figure 1). Each action appears both as a menu item and as a toolbar button. The left hand portion of the screen contains tabbed panes for editing source code. The right hand

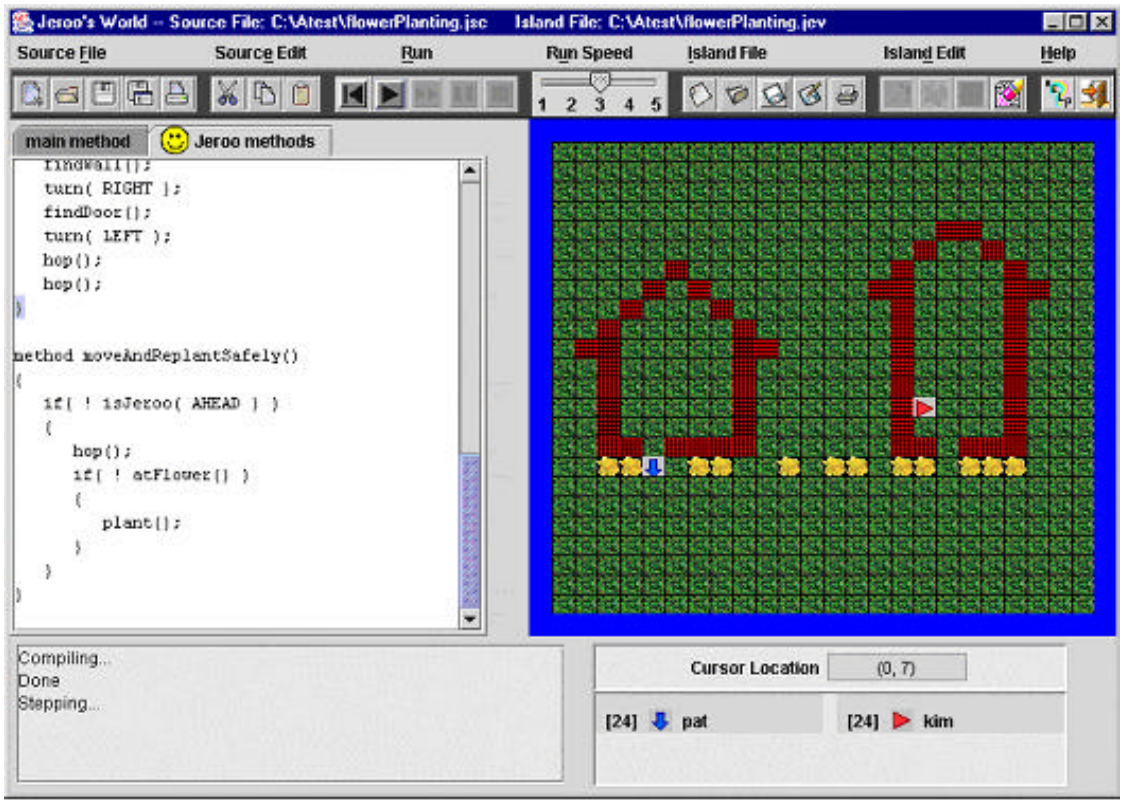

Figure 1 - The User Interface for Jeroo

portion contains a representation of the island. The lower part of the screen is used to display status information.

3.2 Language Features

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.

The Jeroo class has four constructors that allow a student to specify the values of specific 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.

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

Action methods are essentially void methods that 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. These user-defined methods lack arguments, but they can call other methods. The ability to write additional methods is an effective way to introduce modularity at an early stage.

3.3 Editor Features

The Jeroo tool provides a straightforward way to edit both the source code for a Jeroo program and the layout of the island. The source code and island editors support printing and the common file operations: *new*, *open*, *save*, and *saveAs*. The source code editor also supports the common *cut*, *copy*, and *paste* operations.

Source code is entered into a text area consisting of two tabbed panes. One pane is used for the main method. The main method contains statements that instantiate the Jeroos and use them to solve a specific problem. The second pane is used to write the code for the user-defined methods that extend the behavior of the Jeroos. This physical separation of the main method and the Jeroo methods is a subtle introduction to the creation of multiple classes, a topic that will appear later in the course.

Island editing is a point-and-click process. Flowers and nets are added by selecting the appropriate item and left-clicking on the island. Right-clicking on an item will remove it from the island. A location panel helps the students select a specific cell on the island.

3.4 Runtime Features

The Jeroo tool includes a runtime module that illustrates the connection between source code and visible actions. A program may be executed continuously from start to finish, or it may be executed stepwise. When a program is running in either mode, the source code is highlighted, and the actions of the Jeroos are animated.

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 (name), 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. Both syntax and runtime errors can be located quickly; the semantics of the control structures are readily apparent; and the interaction between methods is revealed.

4 USING JEROO

Jeroo has been used successfully in a beginning programming course at Northwest Missouri State University. Thus far, four different faculty members have taught that course using Jeroo. Our use of Jeroo is very similar to the way Karel is used at the University of Waterloo [1]. We use Jeroo for the first four weeks of a 14-week trimester. At the end of these four weeks, we expect the students to understand the semantics of basic control structures, to be comfortable with the concept of using objects to solve a problem, to be comfortable with the process of sending of messages to objects, and to be able to write methods that extend the behavior of the Jeroos.

Educators frequently voice two concerns about the use of a tool such as Jeroo. Can we devote four weeks out of a semester to Jeroo and still cover the required topics adequately? Won't the students have difficulty making the transition from Jeroo's language to a "real" language. Our experience with Jeroo provides reassuring answers to these questions.

Those who have used Jessica (Jeroo's predecessor) or one of the Karel-like tools have reported, informally, that they can cover the same material as before. Our work with Jeroo demonstrated that we can cover the same material without rushing and with no loss of comprehension. We just cover the material in a different order.

The transition to a "real" programming language has been a difficulty for students who used Jessica, or the various Karel-like tools. Our experiences with Jeroo are quite different. The transition to Java has been seamless for the students. We expect that a transition to C++ would be equally uneventful. We feel that this is due to the care that was taken in the design of the Jeroo language, and to the fact that the instructors make continual comparisons between Java and the Jeroo language.

5 PERCEIVED BENEFITS

We have observed several benefits associated with our use of Jeroo. These benefits fall into two broad categories: programming concepts, and student satisfaction.

From the perspective of programming concepts, we feel that the students have a better understanding of control structures, methods, and objects. The early introduction of control structures allows earlier and more frequent use than in our former approach to the course.

Student satisfaction was assessed via a questionnaire that was given to all students near the end of the sixth week of the semester. This date was chosen so that the students could

evaluate Jeroo after they had made the transition to Java. The questionnaire solicited demographic information, and asked the students to use five-point Lickert scales (1=low, 5=high) to answer to the following five questions.

1. Rate your confidence in your ability to learn computer programming before you started the course.
2. Rate your confidence in your ability to learn computer programming after working with Jeroo.
3. Rate your level of comfort about this course before the semester began.
4. Rate your level of comfort about this course after working with Jeroo.
5. Do you think that working with Jeroo was worthwhile to you?
6. Do you think that working with Jeroo was worthwhile to your classmates?
7. Do you think that working with Jeroo was a good way to lead into Java?

Fifty-two students returned completed questionnaires, but one neglected to report a gender. The data are summarized in figure 2.

	Counts	Mean Values				
		Increase in Confidence	Increase in Comfort	Value to Student	Value to Classmates	Value as Introduction
Overall	52	0.5	0.6	3.7	3.8	4.0
Year in School						
Freshman	19	0.3	0.2	3.4	3.7	3.7
Sophomore	10	0.4	0.8	4.2	4.2	4.5
Junior	13	0.8	0.9	3.9	3.7	4.2
Senior	7	0.6	0.7	3.6	4.1	3.8
Other	3	1.7	1.5	3.8	3.7	3.7
Gender						
Female	12	1.0	1.4	4.4	4.2	4.5
Male	39	0.4	0.4	3.5	3.7	3.8
Prior Programming Experience						
No prior programming experience	21	1.0	0.9	3.9	3.8	4.2
Exposure to programming in another course	14	0.6	0.8	4.0	3.9	4.0
Completed a programming course	13	0.1	0.3	3.2	3.6	3.7
Self-Taught programming	3	0.0	0.7	3.9	4.3	4.0

Figure 2 Summary of Questionnaire Responses

These data show that all groups of students felt that working with Jeroo was valuable to them, valuable to their classmates, and valuable as an introduction to Java. As we expected, those who had completed a prior programming course found less value in Jeroo, but even those students rated its value above the middle of the scale. The one unexpected result was that the female students rated Jeroo's value significantly higher than did the males.

Using a paired t-test to compare confidence levels before and after, the probability that the mean confidence level actually stayed the same is less than 3.2×10^{-6}. Thus, the average confidence level increased significantly after using Jeroo. Similarly, a t-test on the levels of comfort reveals that the probability of the two means being the same is approximately 5.0×10^{-6}. We again conclude that using the Jeroo tool has significantly raised the average comfort level of students.

Other interesting conclusions result from several two-sample t-tests. At a statistical confidence level of 90%, the data also indicate that working with Jeroo produced significantly smaller increases in confidence and comfort among freshmen than among students with more years of college experience. This can be attributed to the fact that the freshmen had the highest levels of initial confidence and comfort, leaving less room for increase. As one would expect, the data indicate with 99.92% certainty that working with Jeroo led to greater increases in levels of confidence and comfort among those with less programming experience.

The one surprise is that working with Jeroo led to markedly greater increases in confidence and comfort among female students than among males. The initial confidence levels were essentially the same for males and females. The initial comfort levels were lower for females, but the post-Jeroo comfort levels were lower for males. One possible explanation for this difference is that females made up a larger percentage of the students with no programming background (33%) than any of the other three experience levels.

6 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 [4,5] 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]. The Waterloo version was created as a Java package that can be imported into a standard Java program. The following subsections summarize some important differences between the Jeroo tool and Karel's descendants.

6.1 The Programming Language

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

6.2 The Development Environment

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

6.3 Karel's World Versus Jeroo's Island

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.

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.

6.4 Iteration Structures

Karel's language supports both a while loop and a counter-controlled loop. Jeroo's language only supports the while loop. There were two reasons for this decision. First, our experiences with the Jessica tool indicated that the counter-controlled loop was rarely used. Second, we would need variables to emulate the syntax of Java and C++.

6.5 Conditions

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

6.6 Advanced Features

Karel's descendants tend to require that students write new classes to extend a basic robot class. They also include features such as data types, variables, concurrency, and recursion.

Keeping our audience and intended use in mind, we chose not to require that students write classes. Instead, we allow them to write additional methods to extend the behavior of all Jeroos. Classes come later when we study Java per se. Our experience with Jessica indicated

that could meet our education objectives without data types and variables. We eliminated concurrency because the non-deterministic execution would confuse rather than enlighten the students. The Jeroo language does support recursion, but we do not use it in the course.

7 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 either Java or C++. The use of Jeroo allows us to cover the same topics, but teach them differently and in a different order. Students have favorable impressions of Jeroo, and report increased levels of confidence and comfort after using the tool. If the use of Jeroo is carefully planned within the context of a Java or C++ 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* (February 2001), ACM Press, 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] 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* (February 2001), ACM Press, 16-20.

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

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