

Jeroo: A Tool for Introducing Object-Oriented Programming

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 support a functional decomposition of the task. Jeroo is similar to Karel the Robot and its descendants, but has a narrower scope than Karel's descendants 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. Jeroo and user documentation are available at <http://www.nwmissouri.edu/~sanders/Jeroo/Jeroo.html>.

Categories & Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education – *computer science education*.

D.3.3 [Programming Languages]: Language Constructs and Features – *classes and objects, control structures, procedures functions and subroutines*.

D.1.5 [Programming Techniques]: Object-oriented Programming.

General Terms: Languages

Keywords: Pedagogy, CS1, Objects-First, Microworlds

1 Introduction

The first programming course has always been difficult to teach. When computing curricula emphasized procedural programming, we observed that control structures and subprograms were among the more difficult topics for our students to master. As curricula moved to object-oriented programming, we observed that the list of difficult topics expanded to include the concept of objects.

This paper introduces Jeroo, a tool that helps beginning students master these difficult topics. Jeroo is an integrated development environment and simulator that was inspired by Karel the Robot[7] and its descendants.

In the mid 1980's, 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. Examples were acted out or "animated" on the chalkboard. 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-

Permission to make digital or hand copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, require prior specific permission and/or a fee.
SIGCSE '03, February 19-23, 2003, Reno, Nevada, USA.
Copyright 2003 ACM 1-58113-648-X/03/0002...\$5.00.

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, the senior author of this paper revisited the Jessica project, and worked with students to define the requirements and construct a prototype for an object-oriented successor to Jessica. The result is the Jeroo tool.

2 Design Goals for Jeroo

A few major design goals guided the development of the Jeroo tool.

- Design the tool for novice programmers.
- Keep everything as straightforward as possible.
- Keep everything visible at all times.
- Build a tool to use in just the first part 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.
- Allow sophisticated problems to be solved without complicated code.

Frequently, we referred to these goals to curb our enthusiasm for adding more features. The result is an effective tool that is easy to use.

3 Capabilities of Jeroo

The Jeroo tool has three major components. The Jeroo programming language has a basic set of features and a syntax that provides an easy transition to Java or C++. Editors allow a student to develop programs and arrange the initial positions of flowers and nets on the island. The runtime module provides both language translation and visual execution of a Jeroo program. The user interface consists of a single window in which all components are visible at all times.

3.1 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. Integer literals can be used in some of the constructors. 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.

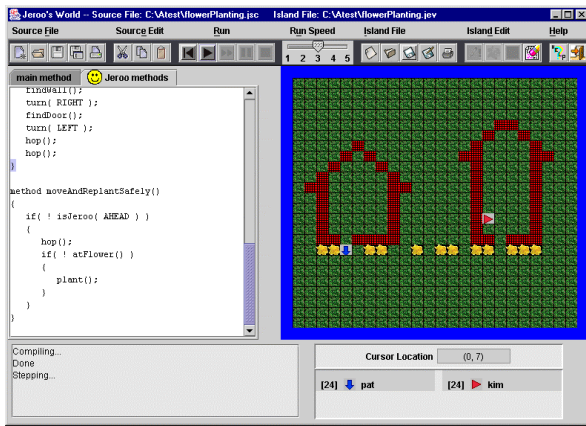


Figure 1 – The User Interface

The Jeroo class has four constructors that allow a student to specify the values of various attributes. These four 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 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. The user-defined Jeroo 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.

3.2 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 to write the code for the main method. The main method contains the code that instantiates the Jeroos and uses them to solve a specific problem. The second pane is used to write the code for 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 essentially 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 student select specific cells on the island.

3.2 Runtime Features

The Jeroo application 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. In either mode, the source code is highlighted as the program is being run, and the actions of the Jeroos are animated simultaneously.

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 in the Classroom

Jeroo has been used successfully in a beginning programming course at Northwest Missouri State University. Our use of Jeroo is very similar to the way Karel is used at the University of Waterloo [1]. Our three-credit-hour course lacks a closed laboratory session, but it is taught in a laboratory classroom with one computer per student. This arrangement allows us to use some class time for “hands-on” activities. 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 messages to objects, and to be able to write methods that extend the behavior of the Jeroos.

4.1 Week 1: Actions and Simple Methods

We have chosen to require that the students use “pencil and paper” for all activities and homework in the first week. This is a change from our previous practice in which we had them copy and run a “Hello” type program. We demonstrate simple programs as we discuss basic concepts, but we keep the students away from the computer so that their attention is on planning and reading computer programs rather than on the technology.

By the end of the week, the students are able to design, write, and critique simple programs that use one Jeroo. These programs include the Jeroo’s action methods and may include one or two simple user-defined methods. By the end of this week, the students are comfortable with zero-based counting (to identify locations on the island), and are familiar with modularization. We introduce some basic terminology (syntax, object, message, instance/instantiation), but we don’t expect the students to fully understand these terms until later in the course.

4.2 Week 2: Multiple Objects and Repetitions

During the second week, we solidify the material from the first week, paying particular attention to the meanings behind the terminology. We have the students work with multiple Jeroos and

have activities that emphasize three concepts: all Jeroo objects have a common behavior, any user-defined method applies to all Jeroos, and different Jeroos have different attributes (location, direction, flower count, and appearance). In addition to working with multiple Jeroo objects, the students learn to use *while* loops and to write simple conditions using the sensor methods and the *!* operator.

4.3 Week 3: More Control Structures

During this week, we expand our repertoire of control structures to include *if* and *if-else*. We have activities and assignments that emphasize the difference between repetition and selection structures. Now that we have expanded our repertoire, we introduce the concept of nested control structures. We also introduce the notion of defensive programming by using *if* statements to guard against run time errors such as hopping into a net.

4.4 Week 4: More Sophistication

During the fourth week, we expand our use of conditions to include the *&&* and *||* operators. Our examples and assignments become more sophisticated, but sensible modularization keeps the complexity manageable. Our final assignment with Jeroo is a challenging problem that we expect the students to tackle in pairs, but we do allow students to work alone if they choose to do so.

4.5 Weeks 5 – 14: On to Java

By the start of the fifth week, the students are ready to move from Jeroo to Java. The transition from various incarnations of Karel to a “real” programming language has always been difficult. There are three aspects to this difficulty: a change in syntax, a change in development environment, and a conceptual change.

The change in syntax is the most difficult for the students. Despite their proven value as instructional tools, Karel and its descendants all have a syntax that is rooted in Pascal. In designing Jeroo, we took great pains to ensure that the syntax of the Jeroo language was nearly identical to that of both Java and C++. The only difference is that we have used a simplified header line for each method. This language design has proven to be beneficial, because the students view Java as having the same structure as the Jeroo language but with additional capabilities.

Some have tried to eliminate the change in development environment by building Karel as an extension to an existing language [1]. In designing Jeroo, we considered this approach, but chose to develop Jeroo as an independent microworld in which the source code, the island layout, and the runtime behavior are always visible at all times. We feel that this provides a simpler starting point for novice programmers. We have encountered no major difficulties as we move from Jeroo to the BlueJ [6] programming environment that we use for the Java portion of the course.

The conceptual change has been the most difficult for us to solve. The students have become accustomed to writing programs that have a certain visual appeal. We want students to become conversant with data types (*int*, *long*, *float*, *double*, *boolean*) and the *String* class. We also want them to be comfortable with using and writing both value returning methods and methods with formal parameters. Where do we begin?

After working with Jeroo for a few weeks, we feel that we need to avoid going back to console I/O or pseudo-console I/O using dialog boxes as substitutes for prompt-read style input and *println* style output. Currently, we are experimenting with various combinations of graphics packages and packages for creating simple user interfaces. Our goal is to introduce console output during the last two weeks of the course, and to delay keyboard input until the second course.

5 Perceived Benefits

We have observed several benefits associated with our use of Jeroo. These benefits fall into three broad categories: programming concepts, programming practices, 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.

By the end of semester, most students routinely decompose problems and plan solutions before they start writing code; their designs are better. We attribute this to their earlier experiences with Jeroo in which decomposition and planning are more natural activities than they are in text and number processing programs. We have also observed that the students are better at reading and tracing code, but this is due to the fact that we started these activities with Jeroo, which is easier to read, and continued them throughout the semester.

Increased student satisfaction is one of the greatest benefits we have observed. At the end of the Jeroo portion of the course (the first four weeks) the students appear to have more confidence in their own ability than the students had after the first four weeks of our former approach to the course. They end the course with more enthusiasm for programming, but this is probably due to a combination of Jeroo and the revisions to the course that were a consequence of our use of Jeroo.

What was lost when the use of Jeroo became part of the course? We feel that we lost nothing other than the ability to rely on a textbook. We can cover all the same material, we just teach it differently.

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]. This 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. Most of the differences are in the syntax of the programming language. Karel and its descendants have a syntax that is rooted in Pascal. The Jeroo tool has a syntax that mirrors Java and C++.

6.1 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.

6.2 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.

6.3 Syntax for the "main" Method

Jeroo "main"	Karel
method main()	task
{	{
}	}

6.4 Conditions

A condition (test) in Karel's language 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.

6.5 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, and concurrency.

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.

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. If the transition from Jeroo to Java or C++ is planned carefully, 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] Kölling, M. and Rosenberg, J. An Object-Oriented Program Development Environment for the First Programming Course. *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium* (February 1996), ACM Press, 83-87.

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

Appendix A: A Sample Program

A Jeroo starts in the southwest corner facing East. The Jeroo must clear 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 2.

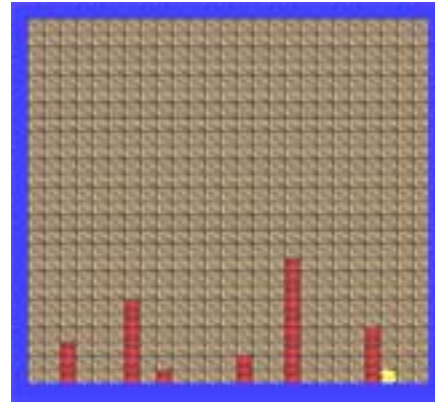


Figure 2

```
method main() {
    Jeroo kim = new Jeroo(25,0);
    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();
}
```