Java meets Karel the Robot

Charlie McDowell
Computer Science Department
University of California
Santa Cruz, CA 95064
charlie@cs.ucsc.edu

Abstract

I have developed a Java implementation of R. Pattis' simulated robot named Karel[1]. The system includes a very simple, easy to use, integrated development environment. It is intended for use in a high school programming course, in a college level computer literacy course, or in the first few weeks of a college level programming course.

1 Introduction

Karel is the name of a simulated robot developed by R. Pattis, and described in his book, "Karel the Robot: A gentle introduction to programming."[1] I learned about Karel a number of years ago, while teaching a computer literacy course, primarily to college freshman. At the time, the course included two weeks of programming to help give the students a better understanding of software. I was using Pascal on Macintosh computers, which included a simple graphics package. Students were guided through creating some simple programs that drew pictures.

I liked the Karel programming environment developed for the Macintosh. It was relatively easy to use. The students didn't have to learn much syntax – the system pretty much handled all of the syntax. They could experiment with conditionals, loops, and procedures (without parameters). There was no concept of a variable, or parameterized procedures.

I never had students in any of my college level classes write Karel programs, but I began showing them a Karel program during the first week of an introductory programming class I was teaching.

The typical introductory programming text describes programming as being similar to a recipe or giving directions. I use an example of programming a robot to fetch the morning newspaper. This leads to a discussion of what instructions the robot already "understands." After developing a "fetch the paper" solution using top-down design, I then make the solution semi-concrete by writing a program for Karel and running the simulation.

In early 2000, I agreed to teach a two-week summer programming course for high school students. I had been thinking about creating a Java implementation of Karel, and this provided the final impetus. The idea was to create a Java class that would allow students to write pure Java code, that was essentially as simple as Karel the robot programs. Because the programs the students write are actually Java programs (or technically part of a larger Java program), the transition to "normal" Java programs is nearly seamless. In addition, the Java based Karel programs can be used to explore any programming concept supported by Java. In particular, students can be introduced to variables, parameterized functions, and OOP.

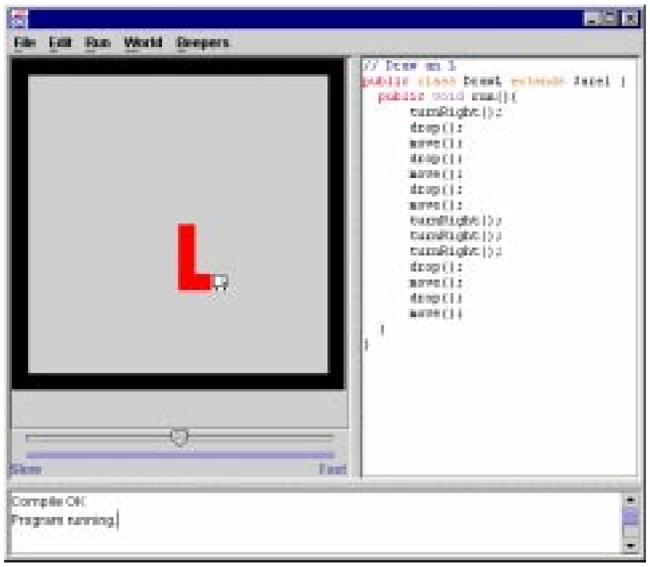
2 What is Jarel?

Jarel is a simulated robot living in a simulated world. Jarel has more or less the same capabilities as Karel. Jarel is implemented as a Java class which the student extends, and supporting classes. The class Jarel (and by extension, the simulated robot Jarel) implements the instructions shown in Table 1.

move	Jarel moves one square in the direction it is currently headed.
turnRight	Jarel turns 90 degrees to the right, remaining in the same square.
pickup	Jarel picks up any beepers that are in the same square as Jarel.
drop	Jarel drops a specified number of beepers into the current square.
clearAhead, clearLeft, clearRight	Jarel checks to see if the square directly in front/left/right is clear (i.e. does not contain a wall or another Jarel).
getBeeperCount	Jarel reports on how many beepers it is currently carrying.

Table 1: Jarel Commands

By default, a Jarel robot is placed in the center of the world facing to the right, with a bag of 1000 beepers. Jarel's world consists of a rectangular grid. Each "move" operation, moves Jarel one grid location. In the original Karel simulation, the beepers were "small plastic cones that emit a quiet beeping noise."[1] Karel can hear a beeper if the beeper is in the same grid location as the robot. Karel can also pick up the beepers, carry them to new locations and put them down. In Jarel's world, the beepers show up in the world by coloring the square – the color depends upon the number of beepers in the square. By moving and dropping beepers, a Jarel program can create simple drawings. A Jarel program that draws the letter L is shown in Figure 1[1].



Compile OK
Program running

```
// Draw an L
public class DrawL extends Jarel {
    public void run() {
      turnRight();
      drop();
      move();
      drop();
      move();
      drop();
      move();
      turnRight();
      turnRight();
      turnRight();
      drop();
      move();
      drop();
      move();
    }
}
```

Figure 1: The Jarel IDE running DrawL.

The first time I used Jarel with high school students, the system was very primitive but still the students seemed to enjoy using it, and learned to program very quickly. In that early version, the supporting environment was a standard Java application that was invoked from a command line. The students edited their programs with a standard text editor, and compiled the programs using javac.

I have now developed a simple integrated development environment for Jarel programs. This simple IDE allows the students to enter/edit Jarel programs, compile them, and run them. A future version of the system may include an integrated debugger. A screen shot from the current system is shown in Figure 1.

[1] Reviewers please note that the code in the figure is repeated in plain text in this version. In the original the text in the screen dump is legible but some clarity was lost in the conversion to PDF. I will attempt to avoid the duplication in the final version.

3 Why Jarel?

One of the challenges in a beginning programming course is to provide intellectually interesting problems early. This is particularly important for younger students getting their first taste of programming. By using a simulated robot the students can solve interesting problems very early on, and seeing your robot "do something" can be very rewarding to the beginning programmer.

I have found that first time programmers find "drawing pictures" very appealing. This opinion appears to be shared by the many textbook authors who have created their own specialized drawing libraries or packages for use with their textbooks. This applies equally to young men and young women. When asked to write a program to make some simple drawing, many students get carried away and create much more elaborate drawings than the assignment called for. This nearly univeral desire to make interesting pictures prompted me to have different numbers of beepers in a square be displayed as different colors. This also leads naturally into a discussion of pixels and color computer displays.

As you can see from the DrawL example Figure 1, a Jarel program requires very little code that the beginning student must include in their program without a full explanation. The method name "run" makes sense (maybe more sense to the novice than "main").

Because `run()` is an instance method, when students begin adding methods to their programs, they don't have to include the hard-to-explain (to novices) keyword `static`.

Because the programs students are writing are, in fact, a Java class, the students can use any Java programming construct or standard Java class. This allows for a seamless transition into "normal" Java programs. At any point the students can be moved off of Jarel into building conventional Java applications.

The important concept of procedural abstraction can be introduced almost immediately - often in the first lecture. Jarel, following the design of Karel, only knows how to turn right. This leads naturally to the creation of a method `turnLeft()`. (In "Karel the Robot" this would be called a new instruction.[1])

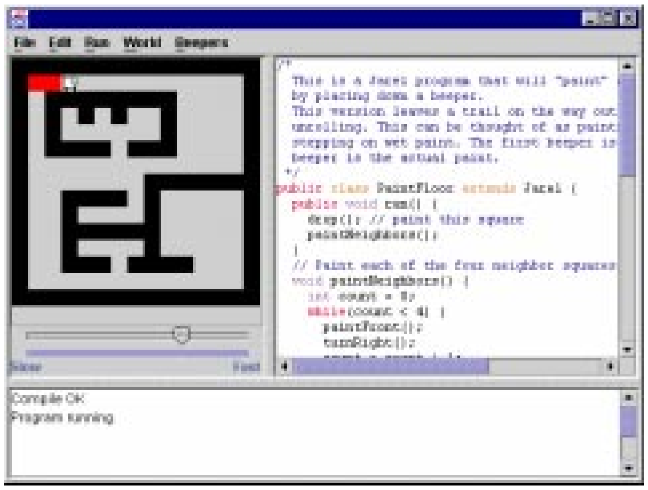
Unlike Karel, with Jarel, the students can also be introduced to parameterized methods. An obvious early example is a method to move X squares. Another limitation of Karel, the lack of variables, has been eliminated with Jarel. Depending upon the course and the instructor, OOP concepts can also be gradually introduced.

4 The Jarel IDE

The current Jarel IDE includes the usual file opening/saving/editing capabilities. Unlike Karel, a Jarel world can contain multiple Jarels. This feature was added to allow for some mild competition. Several Jarels can be placed into a world, each with the same objective (for example, find a particular pile of beepers). The first Jarel to achieve the goal is the winner.

The IDE also allows the user to select a different world. As with Karel, Jarel's world can contain walls. Jarel's walls are rather thick. Each wall element occupies a cell in the world. An example of a Jarel world with some walls is shown in Figure 2.

Finally, you can specify the starting position for the robot, and the number of beepers that the robot is carrying initially. The Jarel in Figure 2 was started in cell (1, 1). The upper left corner is (0, 0).



```
/*
  This is a Jarel program that will "paint"
  by placing down a beeper.
  This version leaves a trail on the way out
  unrolling. This can be thought of as paint
  stepping on wet paint. The first beeper is
  beeper is the actual paint.
 */
public class PaintFloor extends Jarel {
  public void run() {
    drop(); // paint this square
    paintNeighbors();
  }
  // Paint each of the four neighbor squares
  void paintNeighbors() {
    int count = 0;
    while(count < 4) {
      paintFront();
      turnRight();
```

```
Compile OK
Program running
```

Figure 2: A world with some internal walls.

5 Implementation Details

As an experienced Java programmer may have guessed, each Jarel "program" is implemented as a Java thread. To control the speed of simulation, the built-in Jarel instructions (the public methods in the class Jarel) each call `sleep()` after completing whatever else they were supposed to do.

Provided the sleep interval is sufficiently long, multiple Jarels running in the same world will interleave their executions in a simple round-robin schedule with one "Jarel instruction" per time-slice. For the types of settings in which Jarel is intended to be used, this is satisfactory. A scheduler that guaranteed one built-in instruction per time slice could be implemented if desired.

You can obtain a copy of Jarel, including the source, at http:www.cse.ucsc.edu/~charlie/jarel.

6 Conclusion

I believe Jarel provides a gentle introduction to programming with Java and suitable for use with many beginning programming texts. Using Jarel in a beginning programming class has the following advantages:

1. Jarel allows for a very "gentle" introduction to programming concepts ala "Karel the Robot"[1].
2. Because it is built using Java, the Jarel IDE and the resulting Jarel programs are platform independent.
3. The visual feedback provided by the simulated robot is both informative and fun.
4. Everything written for Jarel is done using standard Java syntax, making for a seamless transition to conventional Java applications.
5. Jarel programs can be built using a very simple integrated development environment.

References

1. Richard E. Pattis, *Karel the Robot: A Gentle Introduction to the Art of Programming*, second edition, John Wiley & Sons.