Eric Roberts and Jerry Cain
CS 106J

Handout #5
April 7, 2017

Using JSKarel

This handout describes how to download and run the JavaScript version of Karel that we will be using for the first assignment.

1. Getting started

Step 1. Download Java

Even though this class uses JavaScript rather than Java, the development environments for both JSKarel and SJS are written in Java, which means that you will need to have Java installed on your computer. To do so, follow the steps in Figure 1, which differ slightly depending on whether you are running on a Macintosh or a Windows PC.

Figure 1. Downloading Java

Steps for downloading Java on a Macintosh

1. Go to the CS 106J website at http://cs106j.stanford.edu. Click on the **Software** link and then click on the link **Download** to get the Java SDK installer for Mac.
2. Launch the installer and follow the instructions on the screen.

Steps for downloading Java on a Windows PC

1. Before installing a new version of the Java Runtime Environment (JRE), it is good practice to remove any existing copies first. To do so
   a. Open the control panel by clicking on **Start**, then **Settings**, then **Control Panel**.
   b. Select **Add or Remove Programs** or **Programs and Features**.
   c. From the list of programs you see, uninstall any occurrences of **Java/J2SE Runtime Environment**, **Java SDK**, or **Java Update**. Note that the exact program name may be slightly different or include a version number, but you generally want to remove anything that includes those names. To remove a program, click on the program name to highlight it and click the **Remove** or the **Uninstall** button, or right-click on the program name and pick the **Uninstall** option.
2. Download and install the JRE from the CS 106J website.
   a. Go to the CS 106J website at http://cs106j.stanford.edu. Click on the **Software** link. Go to the section entitled "Installing Eclipse in Windows" and go down to Step 2. There you will see two links (for the 32-bit version or 64-bit version) of the JRE. You should click the version appropriate for your version of Windows. After clicking this link a prompt will likely appear to ask you whether you want to **Run** or **Save** the file. Click **Run** to begin the download and installation process.
   b. The Java JRE installation program should begin. Do a **Typical** installation, and follow the rest of the instructions given in order to complete your installation.

Step 2. Download JSKarel

Your next step is to download the **JSKarel** application. Go to http://cs106j.stanford.edu and click on the **Software** link, which will bring up a directory containing two ZIP files, one for Mac OS and one for Windows. Click on the appropriate file, which will download it to your computer and unzip the contents in your downloads directory. In either case, you should see an application icon for JSKarel that you can then move to your desktop.

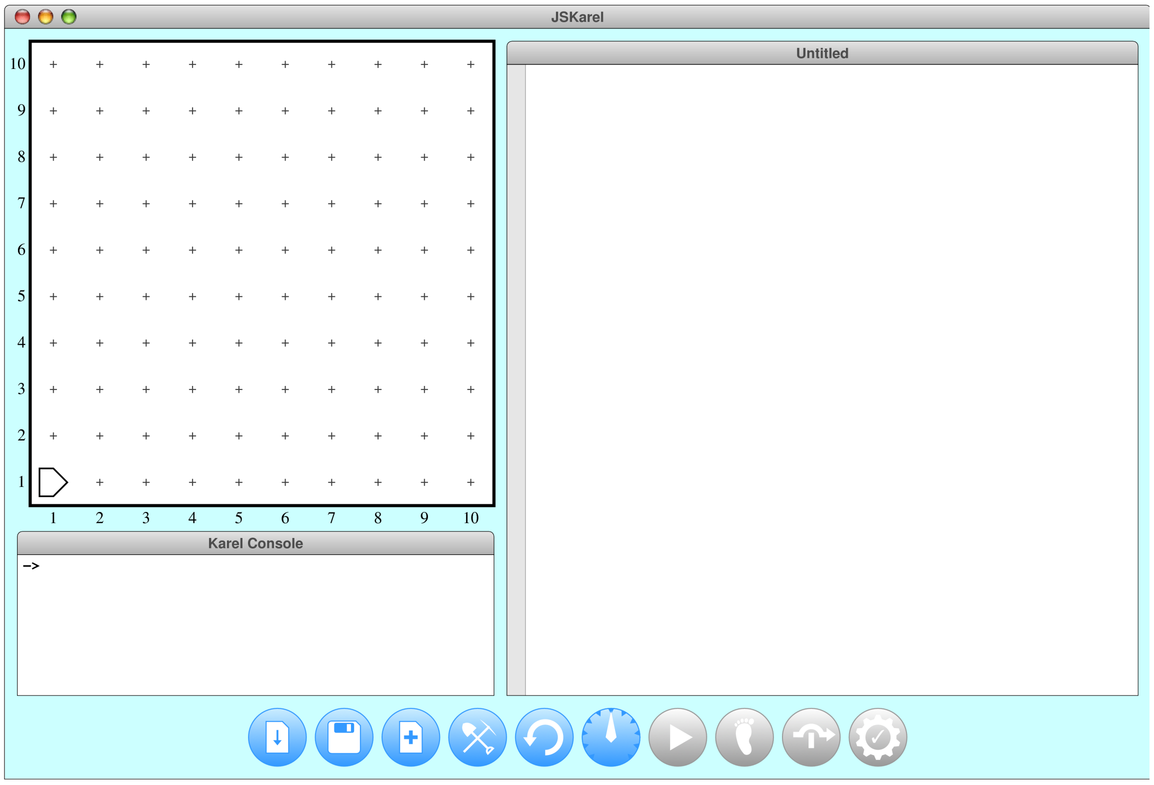
Step 3. Download the starter folder for the assignment

The first step in working with any assignment in CS 106J is to download the starter project for that assignment. Go to the course web page, click on the **Assignments** link, and then choose the entry for the current assignment. For this assignment, you need to click on **Assignment1.zip**, which will download and unzip a folder called **Assignment1** that contains four subfolders, one for each of the four programs on this assignment: **01-CollectNewspaper**, **02-RepairTheQuad**, **03-Checkerboard**, and **04-FindMidpoint**. Each of these folders contains an unfinished Karel program file along with one or more world files that allow you to test your programs in different situations.

Running the interpreter

Once you have finished downloading the starter folder, you're ready to start programming. Double-click on the JSKarel application icon to start the interpreter, which will display the window shown in Figure 2, which is divided into four parts. In the upper left, you see Karel's ***world viewer,*** which allows you to watch Karel's progress. Just under the world

Figure 2. The JSKarel interpreter



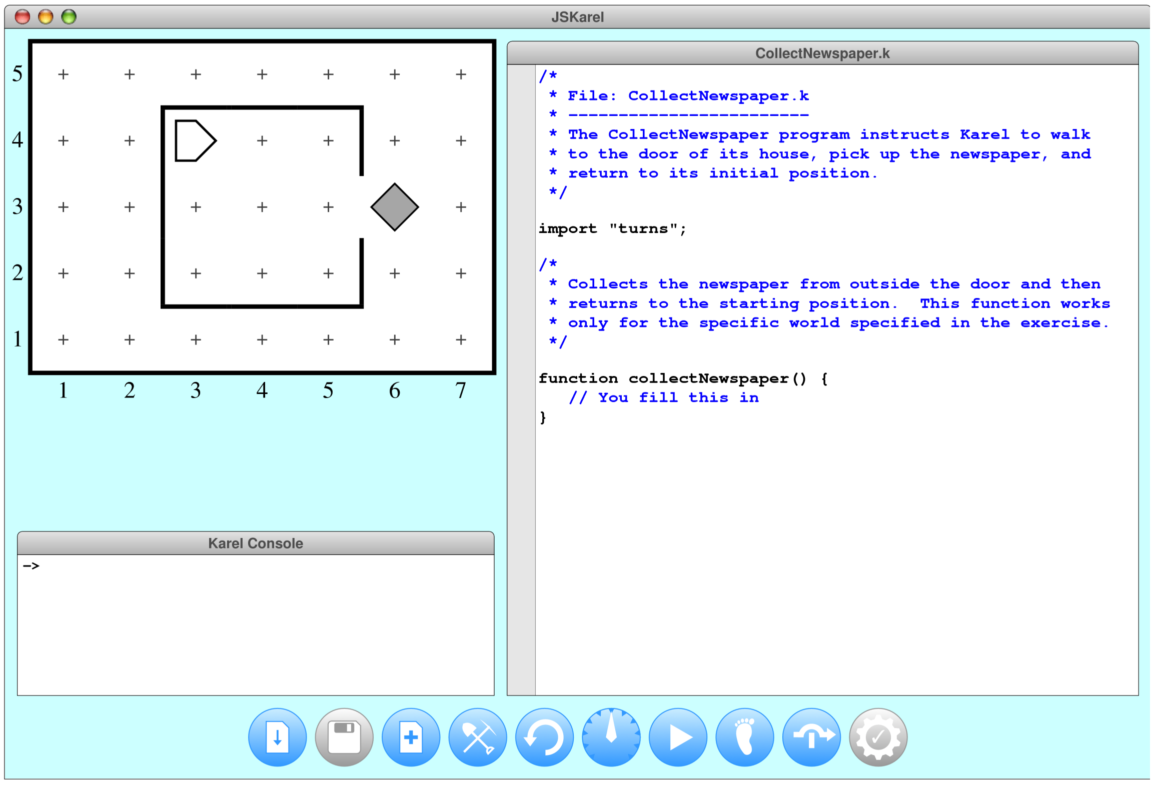
viewer is the ***console window,*** which allows you to type in function calls and watch how they affect Karel's world. The right side of the application contains the ***editor window,*** which is where you enter new programs and edit existing ones. Finally, the bottom of the application contains the ***control strip,*** which contains a set of icons that allow you to control the operation of the interpreter. Each of these icons is explained in a section at some point in this handout.

5. Load the starter file for the program you want to work on

The button at the left of the control strip is the **Load** button, which brings up a dialog box that allows you to select a program to edit. Use the dialog box to navigate through the assignment folder until you find the file containing the starter code for the problem you're trying to solve. For example, if you decide wisely to start at the beginning of the assignment, you should open the **01-CollectNewspaper** folder and then double-click on the file **CollectNewspaper.k**. Doing so loads that file into the editor window, but also automatically loads the world file **CollectNewspaper.w** because the name of the world matches the name of the program. After loading these files, the application window has the contents shown in Figure 3.

As you might have expected, the file included in the starter project doesn't contain the finished solution. Instead, the body of the `collectNewspaper` function is simply a comment reminding you that you need to fill in the details. If you look at the assignment handout, you'll see that the problem is to get Karel to collect the "newspaper" from outside the door of its "house" as shown in the world viewer in Figure 3.

Figure 3. The JSKarel interpreter after loading the CollectNewspaper program



```
/*
 * File: CollectNewspaper.k
 * ------------------------
 * The CollectNewspaper program instructs Karel to walk
 * to the door of its house, pick up the newspaper, and
 * return to its initial position.
 */

import "turns";

/*
 * Collects the newspaper from outside the door and then
 * returns to the starting position.  This function works
 * only for the specific world specified in the exercise.
 */

function collectNewspaper() {
   // You fill this in
}
```

Suppose that you just start typing—even though the assignment handout advises you to decompose the problem first—and define the `collectNewspaper` function like this:

```
function collectNewspaper() {
   move();
   turnRight();
   move();
   turnLeft;
   move();
   pickBeeper();
}
```

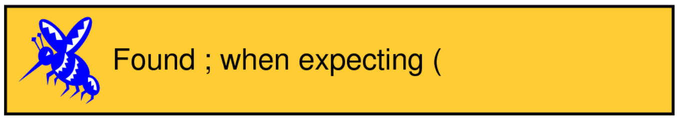
The bug symbol off to the side lets you know that this program isn't going to do exactly what you want, but it is still interesting to see what happens if you try to run the program in this buggy form.

Compiling your program

Before you can run your program, the JSKarel interpreter needs to process the program to make sure that it follows the syntactic rules of the language. This process of checking a program and translating it into an executable form is called ***compilation.*** The **Compile** button at the right end of the control strip triggers the compilation process. In this case, clicking **Compile**, discovers that there is a problem with the line

```
turnLeft;
```

The editor highlights this line and pops up the following error dialog:


Found ; when expecting (

In this case, the error message guides you to the source of the problem, which is that the statement is missing the parentheses after `turnLeft`. This type of error is called a ***syntax error*** because you have done something that violates the syntactic rules of JavaScript. Syntax errors are usually easy to discover because the JSKarel interpreter finds them for you. You can then go back and add the missing parentheses, at which point clicking **Compile** shows no errors.

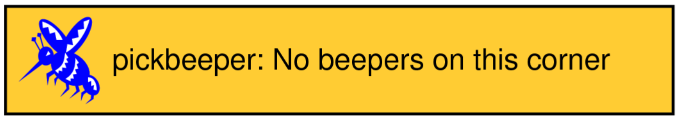
Saving your program

Before you try to run your program, it makes sense to save the file so that your changes are recorded in the file system. The **Save** button does just that. In this case, clicking **Save** writes the updated version of **CollectNewspaper.k**. If you create a new file, clicking **Save** brings up a file dialog and lets you choose the name of the file.

Given that the JSKarel interpreter is new and experimental, it may crash from time to time. You should save your files often to avoid losing your work.

Running your program

There are two ways to run a program after it compiles successfully. The first is to use the console window to enter the name of the main function, followed by the empty parentheses that indicate a function call. You can, however, achieve the same result by clicking the **Run** button, which automatically enters on the console the name of the first function in the program file. If you do that with the current version of the `collectNewspaper` function, things seem to go well for a few steps. Unfortunately, given the program as it stands, Karel ends up one step short of the beeper. When Karel then executes the `pickBeeper` command at the end of the function body, there is no beeper to collect. As a result, Karel stops and displays an error dialog that looks like this:



pickbeeper: No beepers on this corner

This is an example of a ***logic error,*** which is one in which you have correctly followed the syntactic rules of the language but nonetheless have written a program that does not correctly solve the problem. Unlike syntax errors, the compiler offers relatively little help for logic errors. The program you've written is perfectly legal. It just doesn't do the right thing.

Debugging

> *"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."*
>
> —Maurice Wilkes, 1979

More often than not, the programs that you write will not work exactly as you planned and will instead act in some mysterious way. In all likelihood, the program is doing precisely what you told it to. The problem is that what you told it to do wasn't correct. Programs that fail to give correct results because of some logical failure on the part of the programmer are said to have ***bugs;*** the process of getting rid of those bugs is called ***debugging.***

Debugging is a skill that comes only with practice. Even so, it is never too early to learn the most important rule about debugging:

> *In trying to find a program bug, it is far more important to understand what your program <u>is</u> doing than to understand what it <u>isn't</u> doing.*

Most people who come upon a problem in their code go back to the original problem and try to figure out why their program isn't doing what they wanted. Such an approach can be helpful in some cases, but it is more likely that this kind of thinking will make you blind to the real problem. If you make an unwarranted assumption the first time around, you may make it again, and be left in the position that you can't see any reason why your program isn't doing the right thing.

When you reach this point, it often helps to try a different approach. Your program is doing *something*. Forget entirely for the moment what it was supposed to be doing, and figure out exactly what is happening. Figuring out what a wayward program is doing tends to be a relatively easy task, mostly because you have the computer right there in front of you.

Controlling the speed of your program

Often, you can get a great deal of information about what your program is doing just by watching it run. At some point, your program goes off track, which is often easy to spot in Karel's world. It may help, however, to slow Karel down so that you can watch it more carefully. The **Speed** button includes a speedometer-like needle that you can drag around in the button to change the speed. If you turn the dial to the left, Karel runs more slowly. If you turn it to the right, it runs more quickly.

Going through your program step by step

An even more useful debugging strategy involves having the Karel interpreter run your program one step at a time so that you can see what it's doing. You can stop the program at a particular line by clicking in the gray area at the left of the editor window. If that line corresponds to a program statement, the Karel editor will place a ***breakpoint*** on that line (you can clear an existing breakpoint by clicking on it again), which forces the interpreter to stop when it hits that line in the program. At that point, you can use either of two tools to step through your program.

Advancing one step

The **Step** button causes the Karel interpreter to advance by a single step. If the current line is one of Karel's primitive commands, Karel simply executes it and waits for the next command. If the current line is a function that you have defined, Karel starts the process of calling that function and then stops again before executing the first line.

Stepping over the highlighted line

The **Step Over** button causes the Karel interpreter to execute the highlighted line. If the current line is one of Karel's primitive commands, this button behaves exactly like the **Step** button. If the current line is a function that you have defined, Karel executes the entire function call before stopping. This feature allows you to execute an entire function at once, which is particularly useful if that function is one that you already know is working, such as `moveToWall`.

Creating new programs

Although you don't need to create any new program files for the assignment, you will need to do so if you enter the Karel contest or just want to write some programs of your own. The **New** button creates an empty file in the editor that you can then edit and save.

Creating and editing worlds

The one other thing you need to know about—particularly if you're planning on entering the Karel contest—is how to create new worlds and edit existing ones. The **Edit World** button brings up editing palette that contains a whole bunch of icons that allow you to edit the current world. Here is a quick tour of the editor controls at your disposal:

- The large square containing a pair of numbers near the right of the palette allows you to specify the size of the world. If you click on this icon, you can type in a new size, which consists of two integers separated by an x. The first integer is the number of avenues; the second integer is the number of streets. Changing the size of the world erases any beepers and interior walls, so you need to set the world size before editing.
- The and buttons allow you to create and remove walls. To create walls, select the **Draw Wall** tool. If you then go to the map and click on the spaces between corners, walls will be created in those spaces. If you later need to remove those walls, you can click on the **Erase Wall** tool and then go back to the map to eliminate the unwanted walls.
- The five beeper-shaped tools allow you change the number of beepers on a square. The empty beeper tool places a single beeper on any corner you select. The tools marked with the **+** and **–** symbols add a beeper or remove one from a corner. The tools marked with **0** and **∞** set the beeper count on a corner to 0 or infinity, respectively. If you select one of these tools and then click on the beeper-bag icon in the tool area, you can adjust the number of beepers in Karel's bag.
- The four Karel-shaped tools allow you to change the direction Karel is facing. If you need to move Karel to a new starting position, click on the Karel in the world view and drag it to some new location. If you need to put beepers down on the corner where Karel is standing, you have to first move Karel to a different corner, adjust the beeper count, and then move Karel back.
- The various colored squares allow you to paint the corners of Karel's world, as described in the Karel Contest handout.
- When you're finished, you can select the **Save World** tool to save the new world in a file. The **Don't Save World** tool marked with the red x returns to the Karel interpreter with the updated world, but does not save it in a file.

These tools should be sufficient for you to create any world you'd like, up to the maximum world size of 50x50. Enjoy!