Nicholas Bowman, Sonja Johnson-Yu, Kylie Jue
CS 106AP

Handout #7
June 21, 2019

Karel Reference Guide

Karel the Robot exists in a rectangular, grid-world where objects are represented by **beepers**. Vertical grid lines are called **avenues** and horizontal grid lines are called **streets**. Karel can stand at any given intersection of an avenue and a street, also called a **corner**, and avenues and streets may be separated by **walls** through which Karel cannot pass.

Figure 1 shows an example world with all of these different features. Note that different Karel worlds may have different numbers of avenues, streets, walls, and beepers.

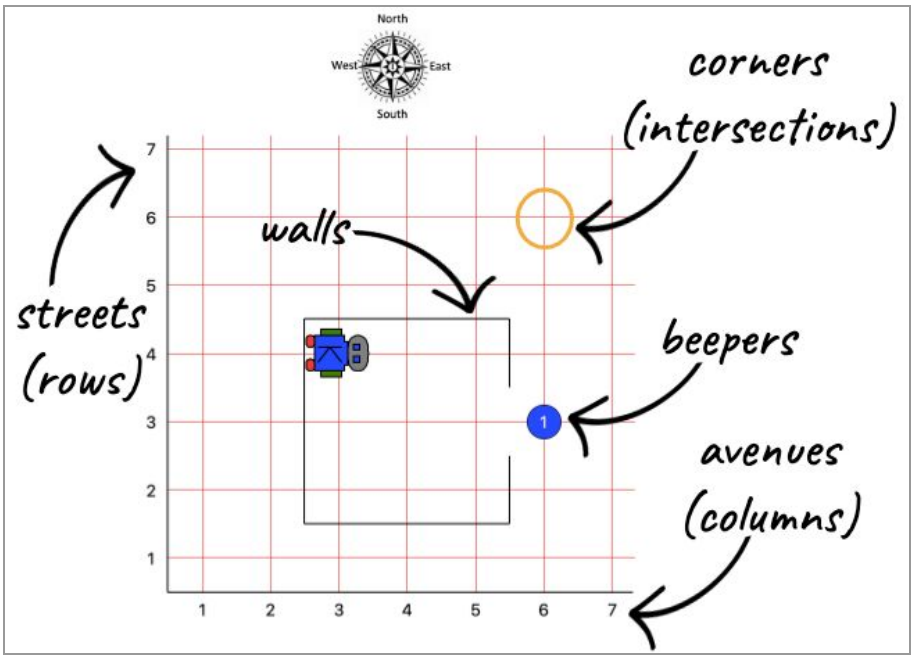

Figure 1: An example Karel world

Built-in Karel commands

`move()`	Karel moves forward one square in the direction it is facing. Karel will crash if there is a wall blocking the way.
`turn_left()`	Karel rotates 90 degrees to the left (counterclockwise).
`put_beeper()`	Karel places a beeper on the corner where it is currently standing. It can place multiple beepers on any given corner.
`pick_beeper()`	Karel picks up a beeper from the corner where it is currently standing. Karel will throw an error if there are no beepers present on the corner.

Figure 2: Built-in commands that Karel already knows

Functions that test Karel's current conditions

Some functions return **`true`** or **`false`** depending on Karel's current conditions, or state, relative to its world.

`front_is_clear()`	Is there no wall in front of Karel?
`left_is_clear()`	Is there no wall to Karel's left?
`right_is_clear()`	Is there no wall to Karel's right?
`on_beeper()`	Is there a beeper on the corner where Karel is standing?
`facing_north()`	Is Karel facing north?
`facing_south()`	Is Karel facing south?
`facing_east()`	Is Karel facing east?
`facing_west()`	Is Karel facing west?

Figure 3: These Karel functions return **`true`** if the answer to the corresponding question is "yes" and **`false`** if the answer is "no."

Creating your own Karel functions

You can also write your own Karel functions to teach Karel how to do new things! To do this, you must use the following structure:

```
def my_function_name():
      """
      Write a good description for your function here.
      """
      # Put Karel commands here
```

Below is an example Karel program with three separate functions, including a **`main()`** function. Notice the comments and how the program is decomposed. You might even consider using the first two functions in your own Karel programs!

```
def turn_around():
      """
      Turns Karel around (180 degrees)
      """
      turn_left()
      turn_left()


def turn_right():
      """
```

```
    Makes Karel turn right (clockwise 90 degrees)
    """
    turn_left()
    turn_left()
    turn_left()


def main():
    """
    Pre-condition: Karel is in the bottom left corner of its
    world, facing east.

    Post-condition: Karel is in the bottom left corner of its
    world, facing north.

    This program places a beeper, moves Karel forward two steps,
    and then makes Karel turn around and go back to pick up the
    beeper. Lastly, Karel faces north by turning right.
    """
    put_beeper()
    move()
    move()
    turn_around()
    move()
    move()
    pick_beeper()
    turn_right()
```