Challenge Lesson: Ozobot Bit—Drunkard's Random North-South-East-West Walk

Created by

Richard Born

Associate Professor Emeritus

Northern Illinois University

richb@rborn.org

Topics

Drunkard's Walk, Computer Science, Algorithm Design, Functions, Procedures, Cartesian coordinate plane, Random numbers

Ages

Grades 7-12

Duration

This should probably be given as a homework assignment for students to do with their own Ozobot Bit at home. It could take several hours to complete, depending upon knowledge of Mode 4 OzoBlocks. The teacher could make the due date a week after assigning the project.The teacher may even with to make this a group project.

Challenge Lesson:
Drunkard's Random North-South-East-West Walk

By Richard Born
Associate Professor Emeritus
Northern Illinois University
rborn@niu.edu

One of the most famous random walk problems involves a drunkard, who with each step, moves one distance unit North, South, East, or West with random, but equal probabilities. You can think of the drunkard as moving about on an infinite two-dimensional Cartesian coordinate plane. If he takes a step north, y is increased by 1; if he steps south, y is decreased by 1. If he takes a step east, x is increased by 1; if he steps west, x is decreased by 1.

Programming Ozobot to do this and keep track of his current coordinates is quite challenging, as Ozobot's innate direction moves are not North, South, East, and West. Rather, they are left, right, straight, and back, as shown in Figure 1.

Figure 1

What we need are four functions (or procedures), and we might name them **goNorth**, **goSouth**, **goEast**, and **goWest**. We would also make use of a variable, maybe called *curDir*, which would keep track of the current direction that *Ozobot Bit's leading edge is facing*. Figure 2 shows a possible way to assign values to the variable *curDir*: 0 if Ozobot is facing north, 1 if he is facing east, 2 if he is facing south, and 3 if he is facing west.

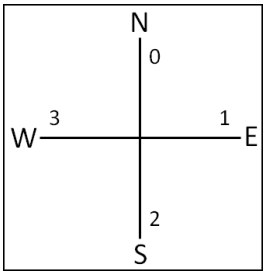


Figure 2

The basic skeleton structure of the random walk then might appear as shown in Figure 3. The four functions **goNorth**, **goEast**, **goSouth**, and **goWest** are randomly called in response to the value of a random number between 1 and 4 inclusive. The job of each of the four functions is then two-fold:

- Depending on the value of *curDir*, pick direction left, right, straight or back, and update the value of *curDir*. For example, suppose that *curDir* is 2 and **goWest** has been called. With a current direction of 2, Ozobot would be facing south. So to go west, he would need to pick direction right and update *curDir* to 3.
- Update the *x* or *y* coordinate of Ozobot Bit. For example, **goNorth** would increase *y* by 1.

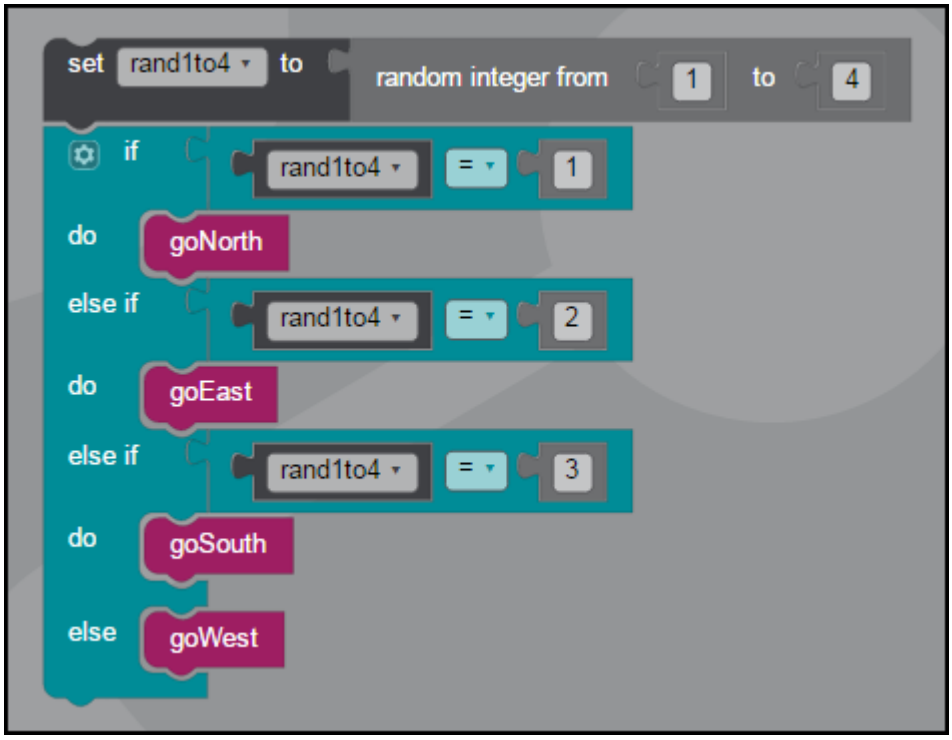


Figure 3

The Map for this Challenge Lesson

Remember that your program is to be written assuming that Ozobot Bit has an infinite Cartesian plane to on which to move. Since an infinite plane is rather impossible for us to make, we will make use of a finite plane as shown in Figure 4. A larger version, suitable for actually running OzoBot Bit, is found at the end of this document. When Ozobot approaches the line ends at the top, bottom, left, and right of this map, your program will cease to run and

Ozobot Bit will stop and blink red and blue quickly. There is another map on the last page of this document. If you use this file to test/run your program, you will need to print a copy of the file and then take the copy to a copy shop and have them enlarge it from 8½" x 11" to 17" x 22", thus doubling the size. The advantage of the 17" x 22" grid is that Ozobot will be able to run your program for a much longer time before reaching the endpoints, stopping, and blinking red. (The cost of having the enlargement made should be in the ballpark of $2.00, as it requires only black and white and not color printing.)

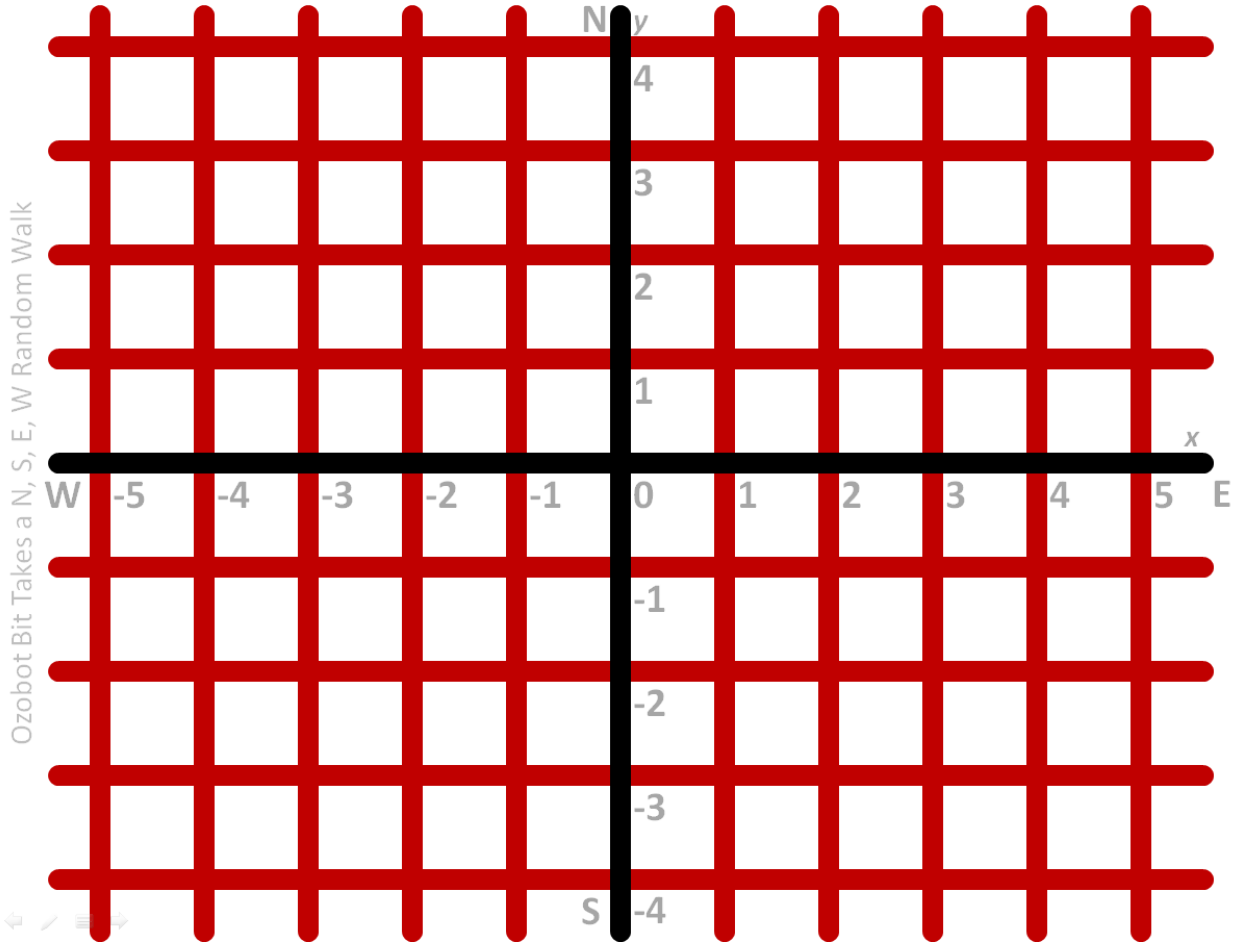


Figure 4

Challenge Requirements

1. Use Mode 4 (Advanced) or higher for programming this challenge. (You may use the "repeat forever" from Mode 3 as well.)
2. Make sure that you calibrate Ozobot Bit on paper before testing and running your program.
3. Make sure that Ozobot Bit's battery is fully charged and the wheels are clean.
4. To start Ozobot Bit's program, you need to double-press the start button.
5. While moving on the Cartesian place, he should display a white LED.

6. Ozobot bit should always be placed on the origin at (0, 0) facing north and then started. Ozobot's first move will be to go north to (0, 1). Therefore, you can initialize *curDir* at 0, *x* at 0, and *y* at 1. From there on, he will make random moves as described earlier. What happens if Ozobot reaches end?
7. Your first goal will be to write and test the main program and four functions **goNorth**, **goEast**, **goSouth**, and **goWest**. In order to determine if your program is working correctly, you need to have Ozobot Bit display a specific color LED for one second, *indicating the direction that Ozobot is going to go next.* Display yellow for north, green for east, dark blue for south, and aqua for west.
8. After successful completion of step 7, disable the code that displays the direction colors. The way to disable a block is to right click on it and then select "Disable Block" from the dropdown. (You can always enable the blocks again later, by selecting "Enable Block".) Now you want to add code to display the coordinates of Ozobot Bit *right after he has made each of his moves*. This code should be in a function, perhaps called **displayCoords**. First display the *x* coordinate. Flash green *x* times if the coordinate is positive, flash red *x* times if it is negative, and show a white LED for a half-second if *x* is zero. Then wait three seconds, and repeat the process for the *y* coordinate.
9. It is rather time-consuming viewing coordinates after every move. Therefore, after successful completion of step 8, modify your program so that the coordinates will be displayed after every 5 steps.
10. If you want to disable showing coordinates all together, you simply need to "Disable Block" on any of your calls to **displayCoords**, as shown in Figure 5.

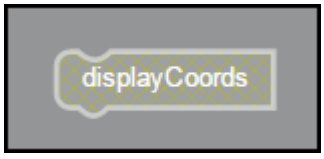
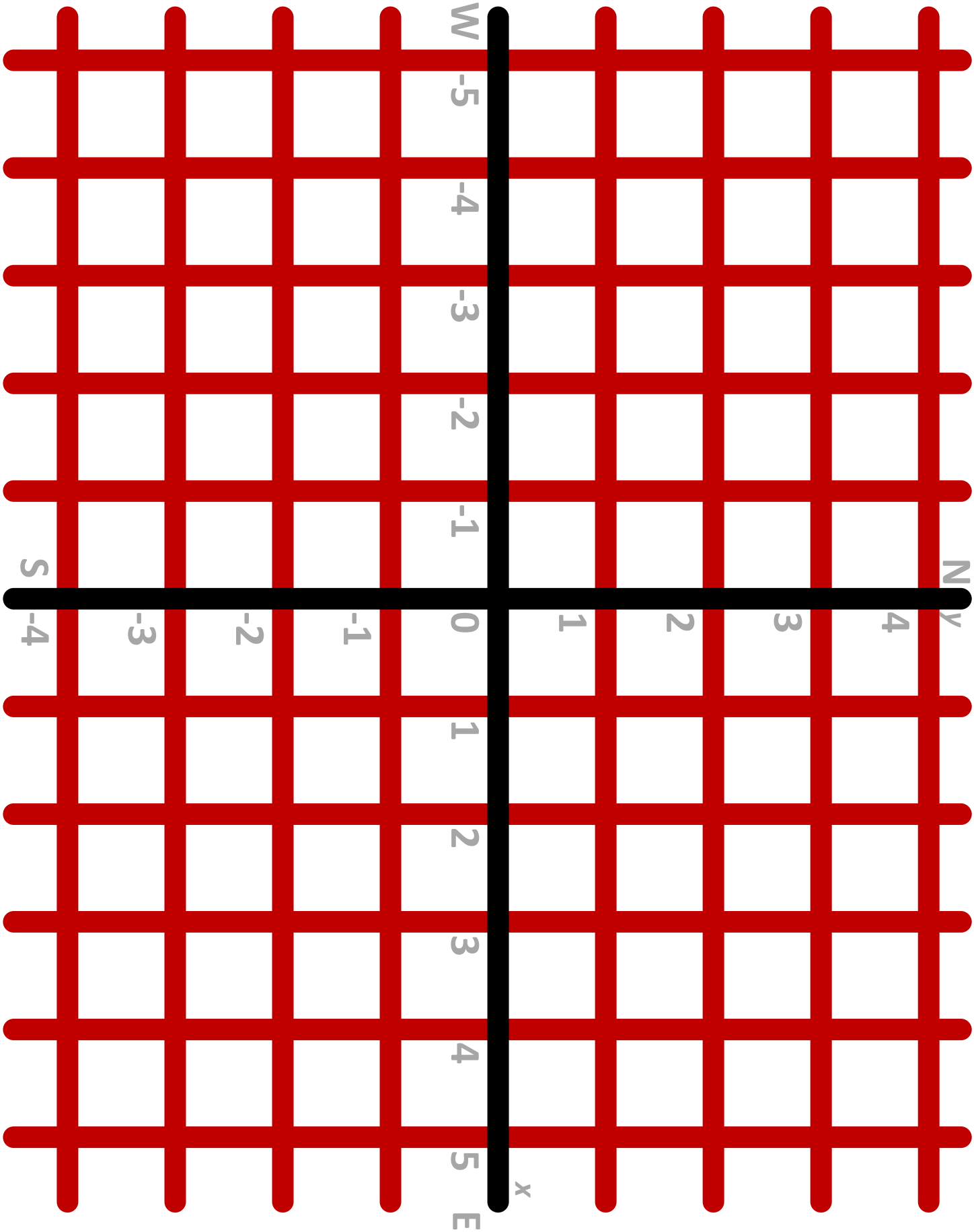
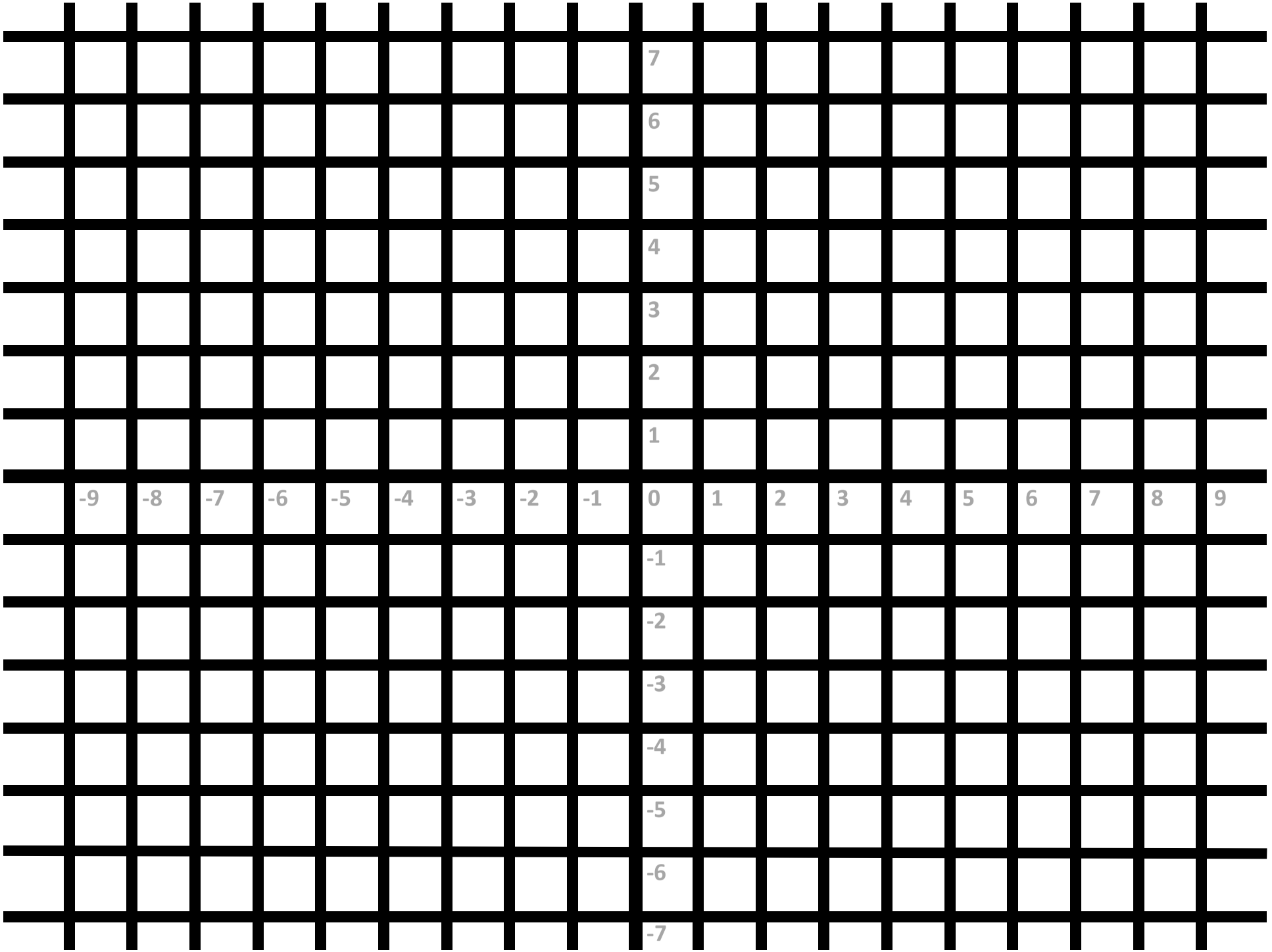


Figure 5

Ozobot Bit Takes a N, S, E, W Random Walk




7
6
5
4
3
2
1
-9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9
-1
-2
-3
-4
-5
-6
-7