Lesson

PATH REVERSAL WITH A STACK

CREATED BY

Richard Born
Associate Professor Emeritus
Northern Illinois University
rborn2@niu.edu

TOPICS

Mathematics, Programming

GRADES

7-12

METHOD

OzoBlockly

DURATION

60 minutes

Path Reversal with a Stack

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

Introduction

In this lesson students will learn how to make Ozobot Bit follow a random path on a Cartesian grid and then follow the path in *reverse direction* back to the starting point. This will be accomplished by the use of a stack constructed from two OzoBlockly Mode 5 (Master) list processing blocks. This lesson assumes no prior knowledge of stacks, so it can be used as a practical and independent learning experience for students.

What is a stack?

A *stack* is a linear list for which insertions and deletions, and usually all accesses, are made from one end of the list. For example, think of a stack of plates on a spring-loaded plate dispenser in a cafeteria. When a customer arrives, he or she removes the top plate from the stack. The process of removing or deleting the top element from a stack is called *popping*. When the stack of plates is low, an employee loads more plates to the top of the stack. The process of adding a new item to the top of a stack is called *pushing*. As a result, a stack can be thought of, alternatively, as a pushdown list. But we'll keep using the term stack since it provides a more visual feeling. At any given time, only the most recently added plate can be removed from the stack. The plates are said to enter and leave the stack of plates in a *last-in-first-out (LIFO)* order.

OzoBlockly Implementation of a Stack

OzoBlockly provides a single list. Two of its list-processing blocks, shown in blue in Figure 1, can be used to construct push and pop subroutines for a stack. Figure 1 also shows the push and pop subroutines along with typical calling blocks.

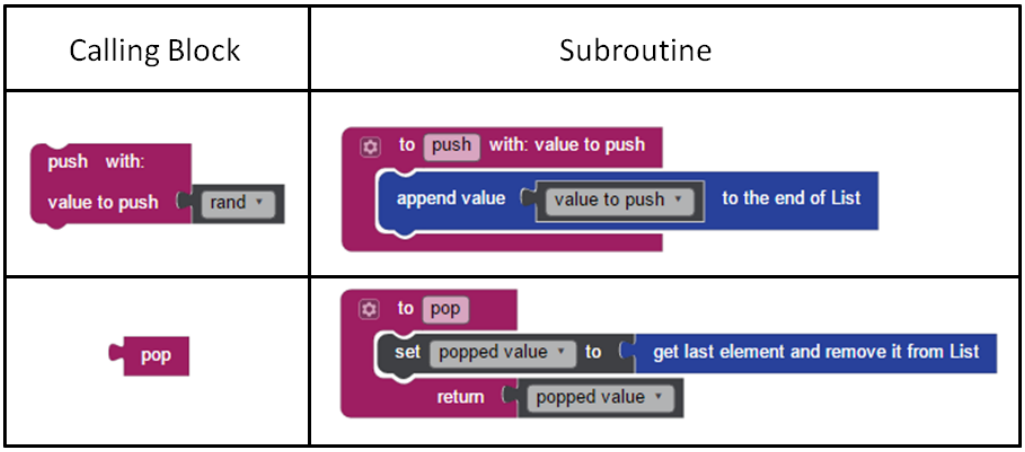

Figure 1

The “append value to the end of the List” block pushes a new value to the end of the list (or to the top of our stack), increasing the length of the list by 1. The example calling block shows that the value of the variable *rand* is being pushed to the top of our stack. As indicated in the OzoBlockly *Master Mode Reference*, the List can hold at most 127 elements, and if the capacity is full, the append operation will fail and the user program will terminate with a blue-red LED animation.

The “get last element and remove it from List” block retrieves and removes the last element in the list (or the top item in our stack), decreasing the length of the list by 1. As indicated in the OzoBlockly *Master Mode Reference*, this block should not be used on an empty list, as doing so will cause a runtime error and terminate the user program. The example calling block is simply a “pop” block for a user-defined subprogram that returns the value popped from the top of the stack.

Bit Reverses Path Video

It is suggested that the video https://goo.gl/T5xBZi be viewed now to clarify what the OzoBlockly program does. You will notice that Bit starts at the origin (gray circle) of the Cartesian coordinate system and is facing to the right. It begins by displaying a white LED. Its initial step is straight ahead to the right. After that, successive steps are randomly either right or left. Bit displays a red LED just before a right turn and a green LED just before a left turn. After completing a total of 10 such steps, Bit displays a blue light, does an about face, and rests for five seconds. Bit then follows the *same path back to the origin* ***but in reverse*** and stops at the origin facing to the right.

For simplicity of program logic, the program was written under the assumption that the Cartesian coordinate system is infinite in all directions. Clearly the coordinate system on the OzoMap is finite. Therefore, you may find that when Bit reaches the outer edge of the finite coordinate system, it stops and flashes the LED red and blue to indicate that it cannot perform a turn requested by the program. In most cases this will not happen if you keep the number of steps in the initial random walk to 10 or fewer. In other words, the greater the number of steps in the random walk, the more likely it is that Bit will reach the edge of the paper’s coordinate system and fail.

The Main OzoBlockly Program

The program was written in a very modularized form with a number of smaller functions (subroutines) that are called as needed in the program logic. The main program simply consists of five functions that are called sequentially. Figure 2 shows the OzoBlockly blocks for this main program.

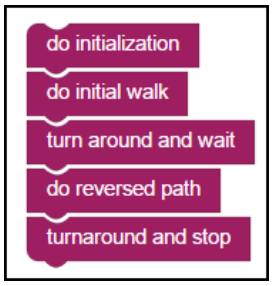

Figure 2

The five functions of the main program are given names that are descriptive of what they accomplish. Basically, they follow what you observed in the video, but we will now discuss each of these subroutines in detail. After that discussion, you should have a good understanding of the entire OzoBlockly program.

"Do Initialization" Subroutine

Figure 3 shows the blocks for the "do initialization" subroutine. Initially the LED is white, the number of steps in the random walk is set to 10, and there is a 2 second delay. This delay allows the student to start Ozobot Bit while holding it and then setting it down on the origin before it starts moving. The line-following speed is then set to a medium speed of 40 mm/s, though this can be set however desired.

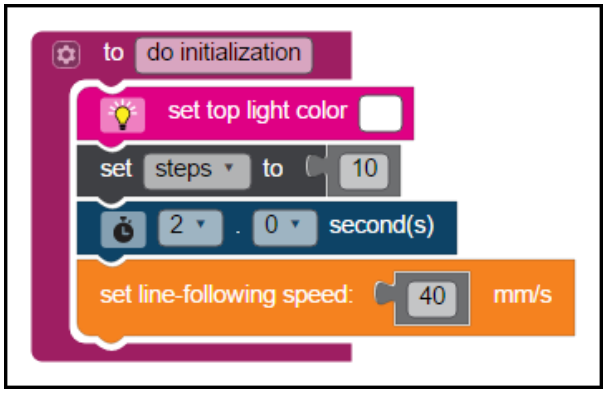

Figure 3

"Do Initial Walk" Subroutine

Figure 4 shows all of the blocks for the "do initial walk" subroutine, the largest subroutine in the program.

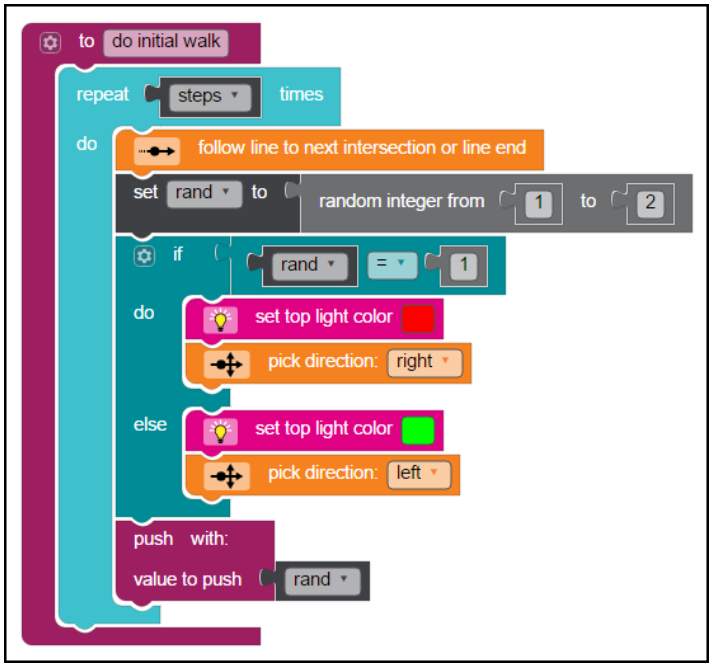

Figure 4

This subroutine contains a loop that is executed once for each step of the random walk. First, Bit is instructed to follow the line to the next intersection. The variable rand is randomly set to either the value 1 or 2. Then an *if* block checks if the value of rand is 1 or 2. If it is 1, the LED is set to the color red and Bit turns right. If rand is 2, the LED is set to the color green and Bit turns left. ***Finally, the value of rand (1 or 2) is pushed to the top of our stack.*** The stack therefore remembers each of the turns in the random walk with the most recent turn at the top of the stack.

"Turn Around and Wait" Subroutine

Figure 5 shows the blocks for the "turn around and wait" subroutine. The LED is set to blue, indicating that the forward random walk is completed. Bit then follows the line to the next intersection and does an about face. Finally, he waits 5 seconds while displaying the blue light to let the viewer know that the random part of the walk is complete.

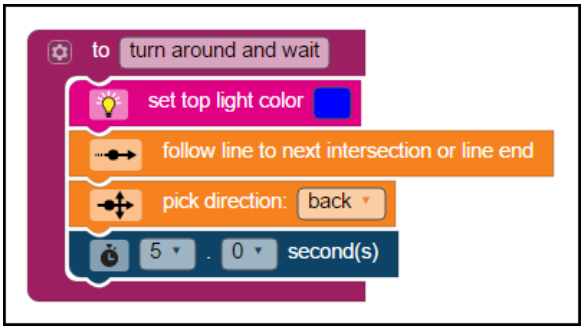

Figure 5

"Do Reversed Path" Subroutine

Figure 6 shows the blocks for the "do reversed path" subroutine. This subroutine contains a loop that is executed once for each of the steps in the reversed walk. Bit is first instructed to follow the line to the next intersection. ***The direction that Bit is to move is popped from the stack as a 1 or a 2***. But *in the reverse path*, Bit must *reverse his direction for every step he takes.* Therefore, if the value is 1, he must make a *left turn* and set the LED to green to indicate he is about to take a left turn. Similarly, if the value retrieved from the stack is 2, he must make a right turn and set the LED to red.

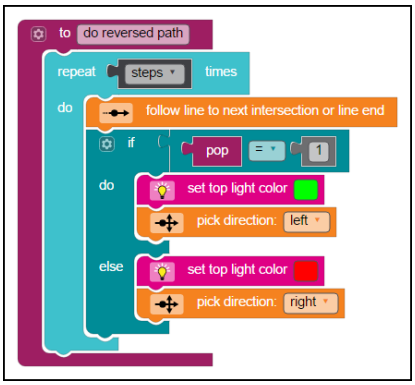

Figure 6

"Turn Around and Stop" Subroutine

Figure 7 shows the blocks in the "turn around and stop" subroutine. In this subroutine, the LED is set to white. Then we want Bit to move to the origin of the Cartesian coordinate system and turn around so that he is facing the same direction as when he started the random walk. This is accomplished by following the line to the next intersection and picking the direction *back*. Finally, the program is terminated and Ozobot is turned off.

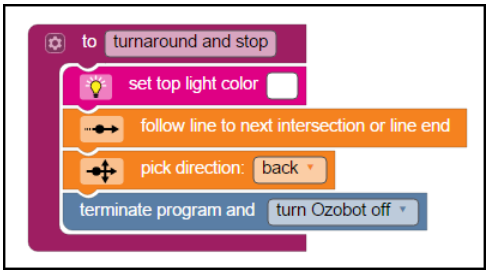

Figure 7

Classroom Activity

After discussing the OzoBlockly program with the class, provide each student or student group with a copy of the *ReversePathWithStack.ozocode* file. Give them a program maintenance task as follows:

> *Users have requested that the program should also allow Bit to move forward during a step, in addition to the existing capability of moving right or left. All three moves should be random and equally probable. While taking a random step forward, Bit should display a yellow LED.*

Optionally, you can tell the students that changes only need to be made in the "do initial walk" and "do reversed path" subroutines. In addition, they need to be especially careful in thinking through the logic that needs to be revised in the "do reversed path" subroutine. *(What is the "pop" subroutine doing, if the "pop" subroutine appears more than one time in the "do reversed path" do loop?)*

The last page of this document contains an OzoMap that can be duplicated for student use with this lesson.