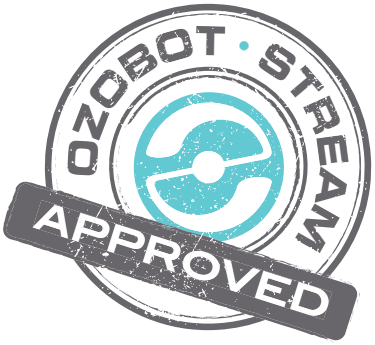

Ozobot Bit Morse Code Generator

Created by

Richard Born

Associate Professor Emeritus

Northern Illinois University

richb@rborn.org

Topics

Computer Science, Functions,

Visual block programming

Ages

Grades 9-12

Duration

Intended as a homework assignment; 1-2 hours

Challenge Lesson:

Ozobot Bit Morse Code Generator

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

Introduction

Most everyone has heard of the international distress signal SOS, "Save Our Souls":

With the letters S and O encoded as a series of dots (dit) and dashes (dah), this signal makes use of what has become known as the Morse code, named after Samuel Morse, inventor of the land-line telegraph system back in 1837. SOS is just one of many abbreviations using Morse code that are understood internationally, making Morse code independent of proficiency in the English language.

Morse code has a number of uses even to this day. It is popular among amateur radio operators but is no longer required for licensing in the United States and many other countries. It is used in the medical sciences as an assistive communication technology for people with severe disabilities, such as being both deaf and blind. A message that can be read by computers is tapped or blinked out by the person with the disability. Skin buzzers can then be used to receive Morse codes. The ability to send distress messages over large distance with relatively low power requirements has made Morse code useful in the military.

Morse code has several advantages in addition to those suggested in the previous paragraph. Morse code can be read by skilled listeners without the need for a decoding device of any kind. Morse code can also be transmitted in a variety of forms including audio tones, radio signals, electrical signals, and visual signals such as a blinking LED. Radio signaling of Morse code is usually in an on-off keyed manner, requiring less complex equipment than other radio communication technologies.

You may be wondering why one would want to develop a Morse code generator for Ozobot Bit. The main educational purpose, as seen by the author of this paper, is that doing so presents a great opportunity for the student to learn hierarchical modular programming by the use of functions. The advantages of such modularity include easier programming, reduction in coding time, avoiding repetition of code, easier maintenance due to the ability to locate and quickly change values of parameters, and programs that require less memory. Reduction in required memory for programs is particularly important when programming small devices with limited memory like Ozobot Bit.

Morse Code Structure

Morse codes exist for letters, numbers, and special characters, and include abbreviations for commonly used words. Figure 1 shows the International Morse codes for letters and numbers.

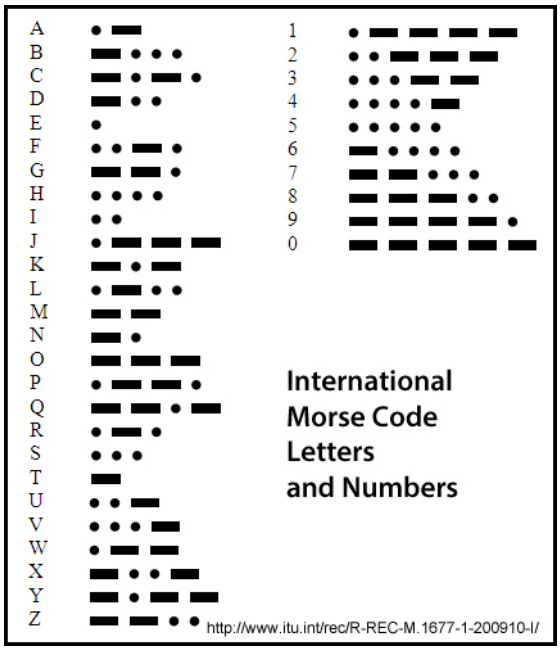

Figure 1

The coding process makes use of a short time gap between elemental parts of the same letter, a longer gap between letters, and an even longer gap between words. Suppose that the length of a dot is one time ***unit***. Then the International Telecommunication Union (ITU) defines the following regarding the spacing and length of signals:

- The length of a dot is one time unit.
- The length of a dash is three time units.
- The space between elemental parts of the same letter is one time unit.
- The space between letters is three units.
- The space between words is seven time units.

With the average length of a word being six letters, the following formula gives a good approximation for the time length, *T*, ***in milliseconds,*** of the time ***unit***, assuming that the goal is the transmittal of *W* words per minute (WPM).

$$T = \frac{1200}{W}$$

For example, 20 WPM would mean that the time unit would be 60 ms.

Functions for the Ozobot Morse Code Program

Figure 2 shows a suggested set of hierarchical user defined functions for programming Ozobot Bit to blink the Morse code. The lowest level functions are in the leftmost column, with progressively higher levels as one moves to the right. An example of a typical main program is shown in the rightmost column, where ***OZOBOT MY TINY ROBOT*** is set to blink 10 times in Morse code. Keep in mind that the suggested functions are based on using OzoBlockly blocks available in "Advanced" mode 4. Capabilities of "Master" mode 5, with plans to support low-level control functions and advanced programming features, are unknown to the author at the writing of this document. These capabilities could possibly result in a different approach to programming Ozobot to produce Morse code.

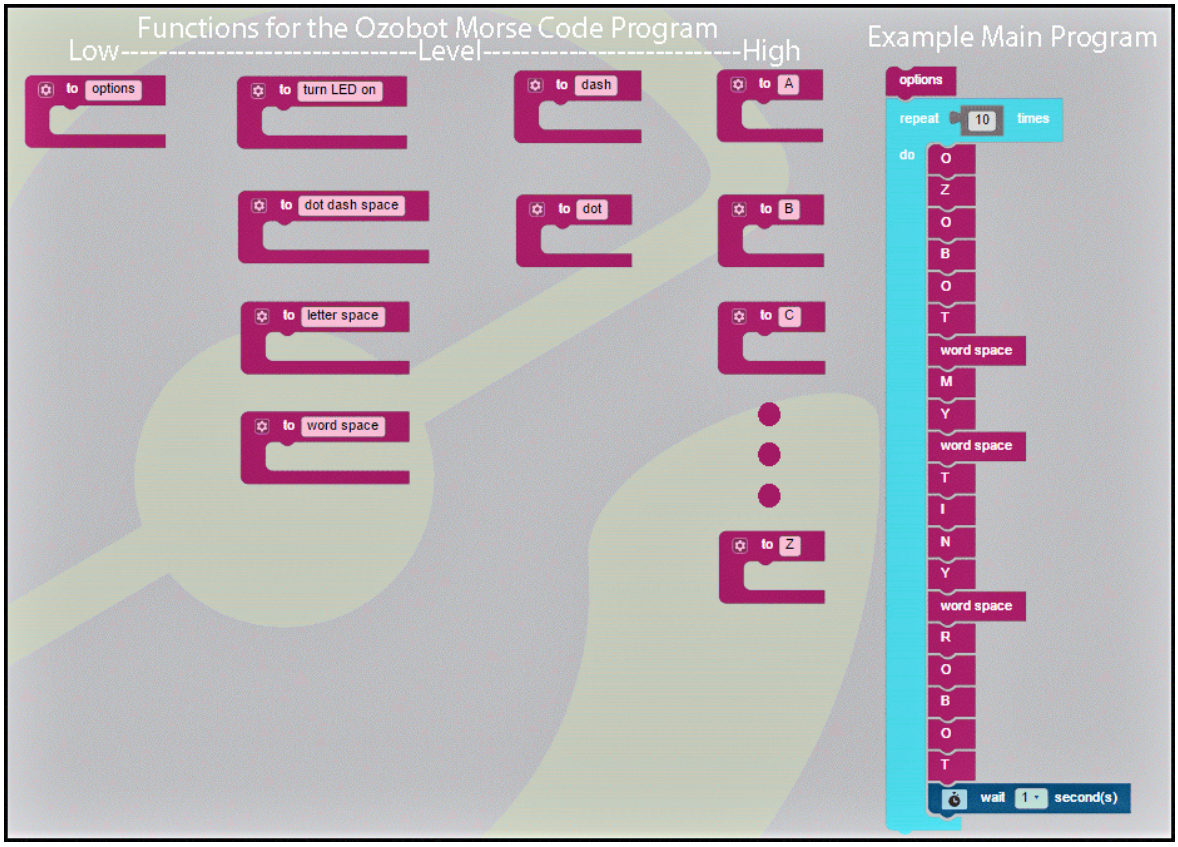

Figure 2

Referring to Figure 2, the lowest level function "options" is where the values of all variables containing numeric constants are declared. This provides a *single location* where adjustments of these values can be made. For example, a beginner who is learning Morse code may want to increase the time between dots and dashes, the time between letters, and the time between words. This also provides a single location where the color that Ozobot blinks its Morse code can be adjusted by setting the red, green, and blue components of the LED's color. Suggested variables with constant values that can be declared:

- Three variables to hold the values for the red, green, and blue components of the LED's color
- Two variables to hold the number of time units for dots and dashes
- Three variables to hold the time unit spacing between dots and dashes, between letters, and between words
- A variable to set the desired words per minute (WPM) for the generated Morse code
- A variable to compute the time length for the basic time unit. Keep in mind that you will be using the OzoBlockly block "***wait … x 10ms***", so you will want to set the time unit to 120/WPM rather than 1200/WPM since the block multiplies x 10 ms.

You may also find it helpful in the options function to initially set the LED to white for three seconds, and then turn it OFF for a second before flashing the Morse code.

Moving on to the next column of functions, first consider the "turn LED on" user defined function. While there is an OzoBlockly function to "turn LED off", there is no OzoBlockly block that turns the LED on with a color that can vary. However, you can turn the LED on by using the OzoBlockly block "set LED color" and use the variables for red, green, and blue that you defined in the options function.

The user-defined functions "dot dash space", letter space, and word space simply turn the LED off and then wait the appropriate number of milliseconds.

Moving on to the next column of user-defined functions, the user-defined functions "to dot" and "to dash", each call "turn LED on", wait the appropriate number of milliseconds, and the turn off the LED.

Moving on to the highest level user defined functions, we see that there are 26 of them, one for each letter of the alphabet. (You probably will not have enough memory in your Ozobot Bit to include the numbers 0 through 9 in your codes.) Each of these 26 letter functions will make use of the two middle levels of user defined functions, according to the definitions of the International Morse code.

That pretty much sets the stage for your programming challenge. Enjoy!

If you'd like to see a short video of the Ozobot Bit Morse code generator in action, visit this link on YouTube:

https://youtu.be/cUc4m261nXg

In this video, Ozobot's blinking LED has been interfaced with a simple littleBits circuit that converts the LED signals from Ozobot to sounds on a speaker. The coded message is ***Ozobot My Tiny Robot***. Note that one could replace the speaker with a vibration motor. This vibration motor could be attached to a deaf and blind person's skin. If this person knows Morse code, he or she could interpret the code through the sense of touch on the skin.