Ozobot Editor—Lesson: Two-Dimensional Arrays—Student Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

This lesson will take you through a sequence of seven Ozobot Editor programs that bring *two-dimensional arrays* to life in a way that will help you understand this data structure. Through the use of a provided Ozomap, Ari or Evo demonstrate array elements by *speaking* the contents of the element while following a line through the array's elements as shown in Figure 1. The elements, memory locations that can store data, are shown in light gray.

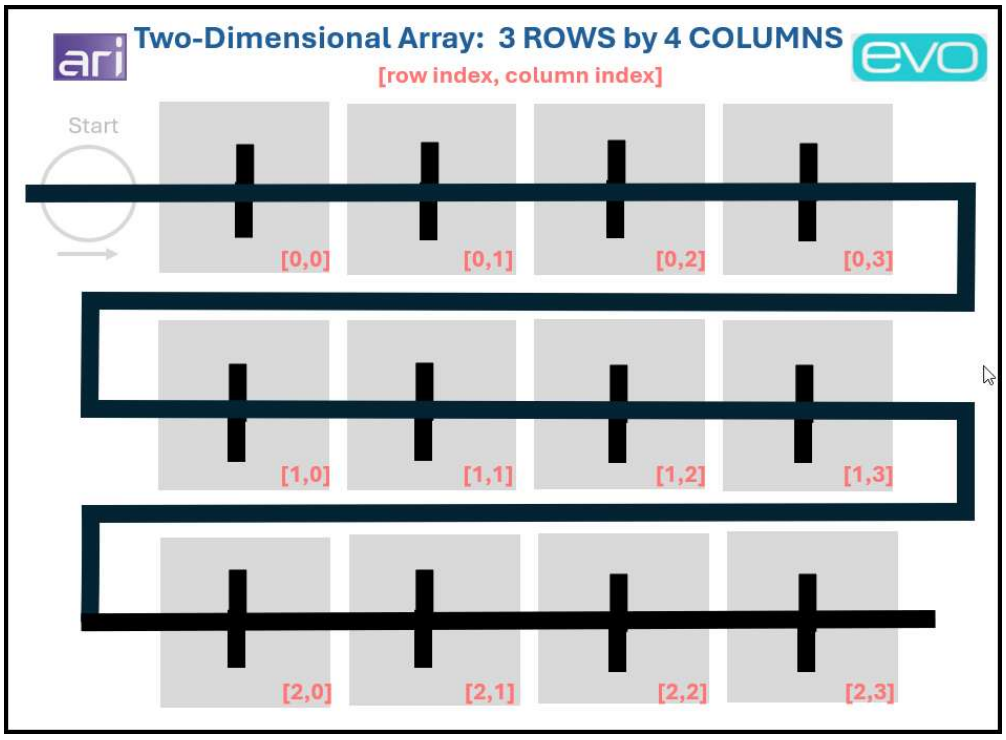


Figure 1

A two-dimensional array can be thought of as a grid or matrix such as the one shown in Figure 1. Rows are horizontal and columns are vertical. Thus, the array shown in Figure 1 has three rows and four columns. The location of a specific element is specified by an ordered pair of numbers called *indexes*. These indexes are *zero-based*. This means, for example, that the first element in any row has a *row index* of 0. Similarly, the first element in any column has a column index of 0. Therefore, each element can be identified by the ordered pair *[row index, column index]*. Note that with arrays, square brackets are used to enclose the indexes. The first item in the square brackets is always the row index, while the second item is the column index. The indexes for each element of the array in Figure 1 are shown in red.

For all the programs in this lesson, Ari or Evo would be placed on the light gray circle labeled *Start* facing the direction shown by the arrow, and then the program would be started by selecting *Run* on the Ozobot Editor screen.

When the Ozobot Editor program *Two-Dimensional Array Template* is opened, you will see the three functions shown in Figure 2. **You need not concern yourself with the details of these functions**.

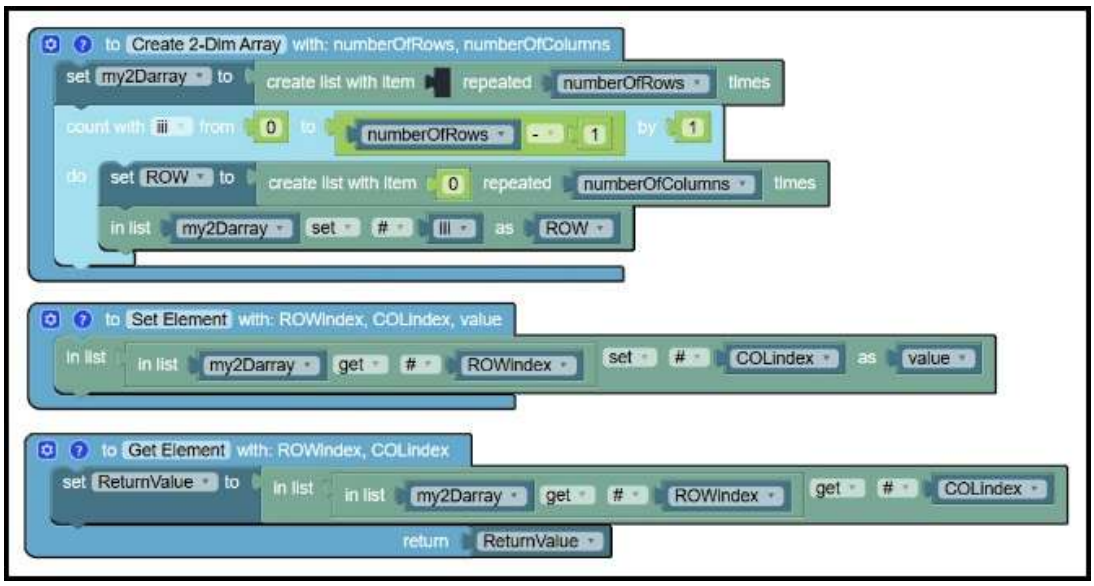


Figure 2

Rather, if you select *Functions* from the list of block types on the far left of the Editor screen, you will see the three function call blocks shown in Figure 3.

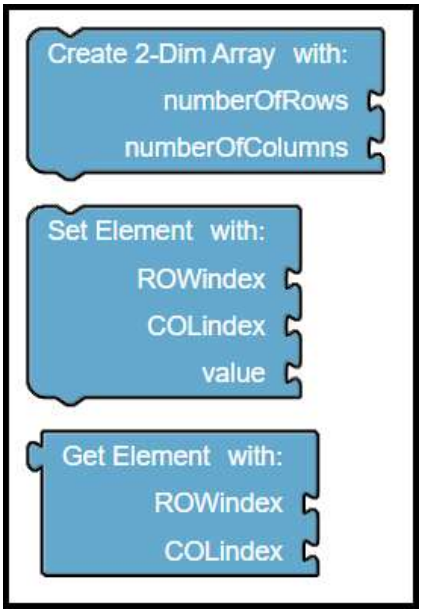


Figure 3

The first of these function call blocks allows you to create a 2-dimensional array by simply specifying the number of rows and columns in the array. In the second function call block, you specify the *row* and *column indexes* of the element whose value you want to set. The final function call block provides a means for getting the value of the element at the specified row and column indexes.

Program 1: Determining the Default Values for an Array Element

Open the Ozobot Editor program *Two-Dimensional Array Template*. Add the blocks shown in Figure 4. The blocks begin with a call to the *Create 2-Dim Array function*. Then there is a double-nested loop, with the outer loop cycling through the row indexes and the inner loop cycling through the column indexes. Notice the -1 in each of these loops—the highest row index is always 1 less than the number of rows, similarly for the columns. Inside the double-nested loops, Ozobot moves to the next intersection, picks the direction straight, and then says the value of the element at that array location. This value is obtained by a call to the *Get Element* function.

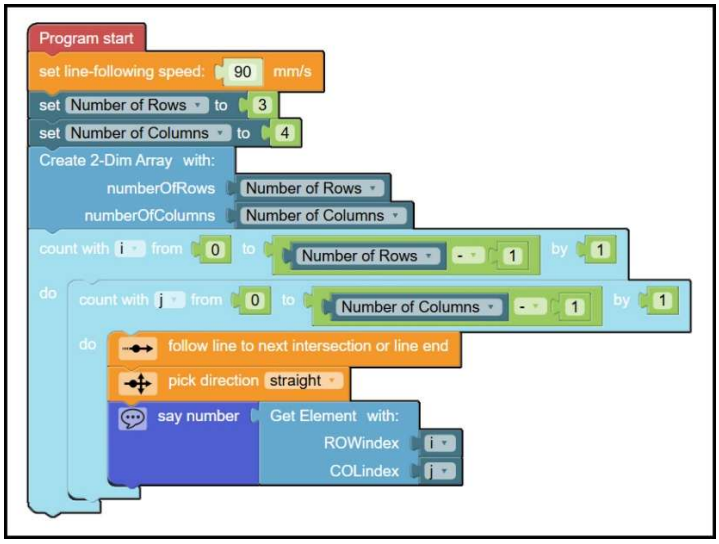
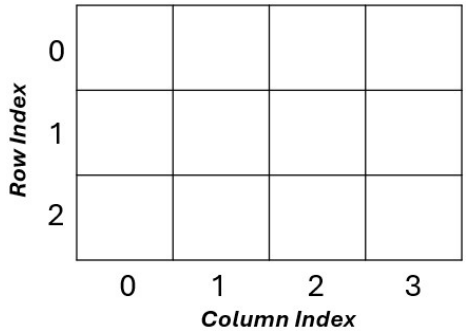


Figure 4

After connecting Ari or Evo, place it on the Ozomap start circle facing the direction shown by the gray arrow. Then select *Run* from the Ozobot Editor screen. Fill in the table below with the values Ozobot speaks while traversing the array.



What do you conclude regarding the default value for each element in your two-dimensional array? This is the value that was specified in the *"set ROW to"* block of the *Create 2-Dim Array* subroutine in Figure 2. There is nothing special about this value—it is up to the software developer to decide issues regarding default values. Once again, let's keep the default value as specified by the author of this lesson at zero—don't modify the subroutines of Figure 2.

Program 2: Filling the Elements of the Array with the Number 5

Now let's modify our program so that each of the elements in the array contains the value 5. To accomplish this, add the double-nested loop calling the *Set Element* function as shown in Figure 5. This function sets the element whose row index is *i* and whose column index is *j* to the value 5, with *i* varying from 0 to 1 less than the number of rows and with *j* varying from 0 to 1 less than the number of columns. All 12 elements of the array are thus set to the value 5.

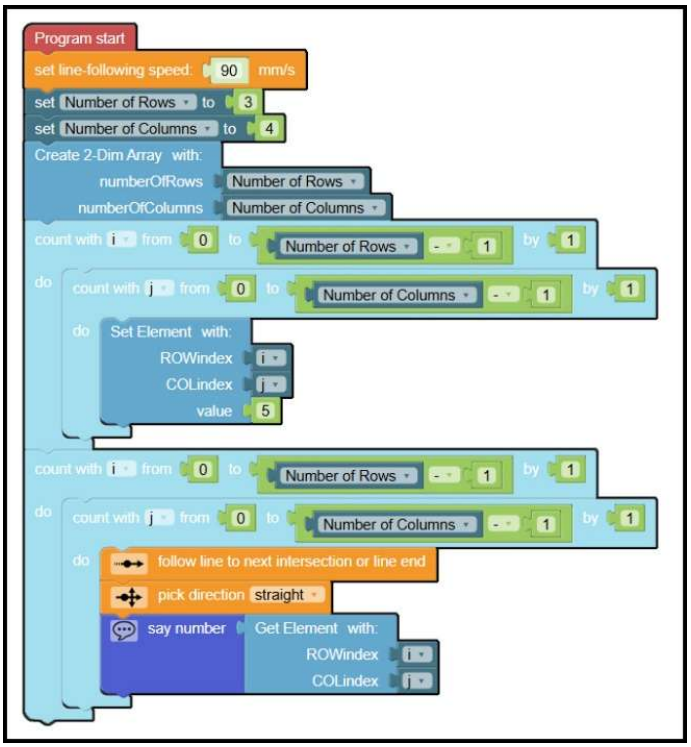
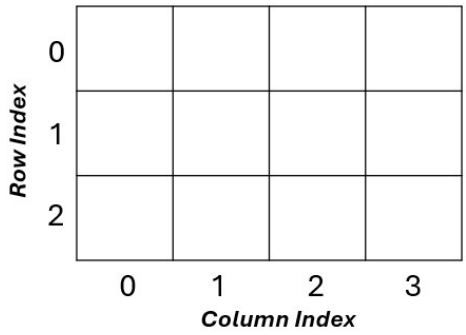


Figure 5

Run the program and record the values that Ozobot speaks for the value of each element of the array in the table below.



This clearly shows that you can change the "default" value of each element to whatever you desire **without** modifying the value in the "*set ROW to"* block of the *Create 2-Dim Array* subroutine in Figure 2

Program 3: Filling the Elements of the Array with a Mathematical Calculation

Figure 6 shows Program 3, in which you will be filling the elements of the array with the result of a mathematical calculation. There is just one change you need to make in Program 2 to get Program 3. It is in the *value* parameter of the *Set Element* function call. It shows the mathematical calculation *i + j*.

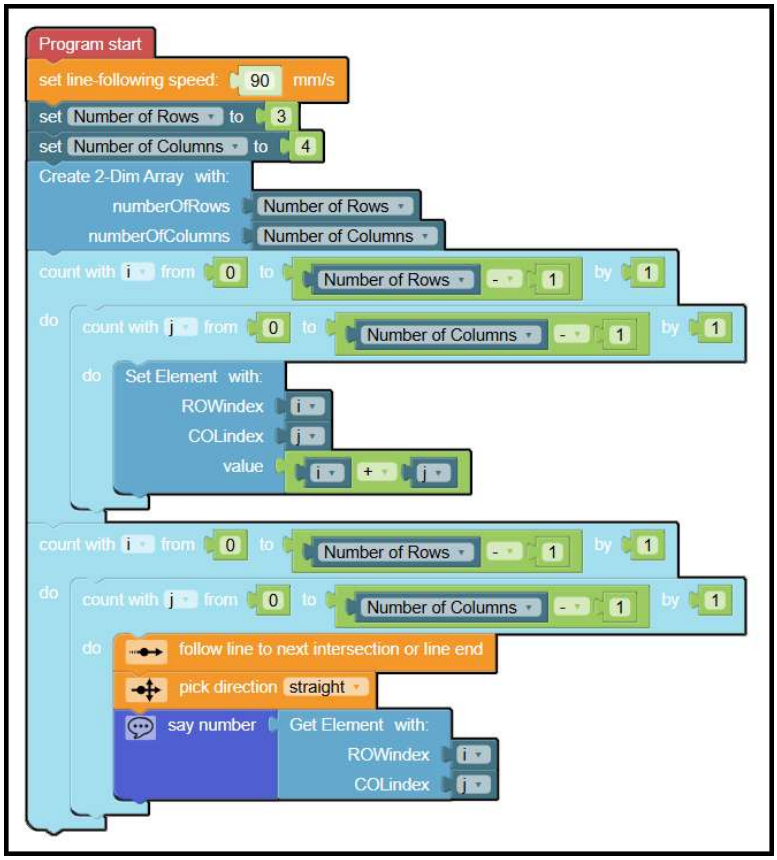
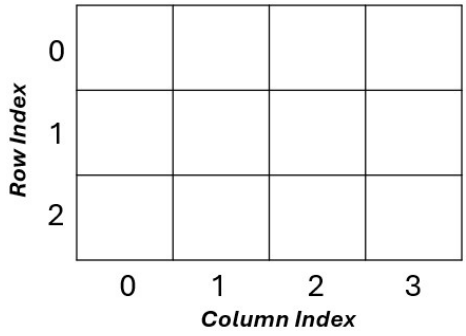


Figure 6

Before running this program, indicate the values that you expect to be in each of the elements of the array in the table below.



Now run the program. Were your predictions correct? State in plain language (not a mathematical formula!) what the value in each cell represents.

*Plain Language*_____

Program 4: Filling the Elements of the Array with a Sequence of Values

In this program, you will fill your two-dimensional array with the squares of the integers between 1 and 12, inclusive, as shown in the table below.

Row Index				
0	1	4	9	16
1	25	36	49	64
2	81	100	121	144
	0	1	2	3

Column Index

The complete program appears in Figure 7. The only changes in this program, as compared to Program 3, include:

- The introduction of a new variable called *sequenceCount,*
- changing the value of this variable by 1 just before the *Set Element* function call, and
- setting the *value* parameter to the square of the sequenceCount.

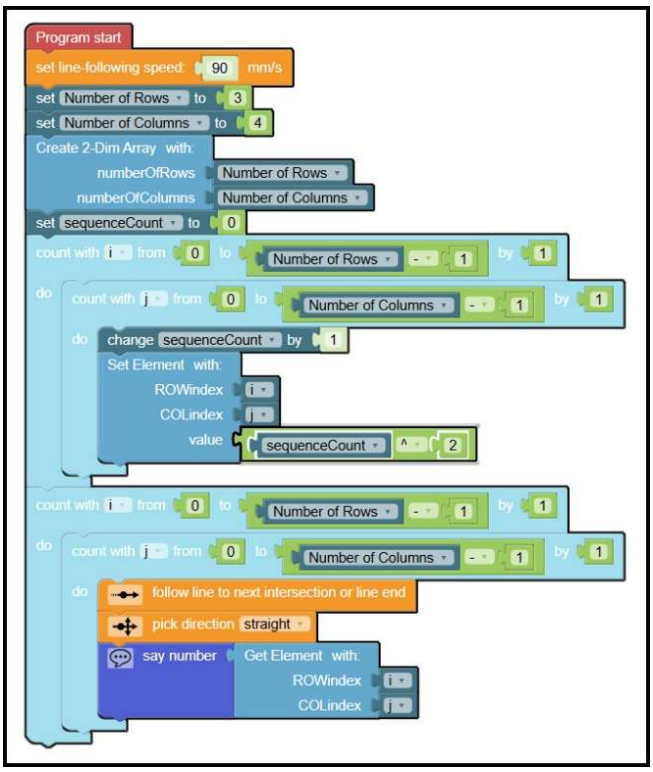


Figure 7

Make these changes to Program 3 and then run the program in the Ozobot Editor with Ozobot starting at the usual location on the Ozomap provided for this lesson. Did Ozobot speak the expected squares of the integers from 1 through 12? If not, debug your code to get it correct.

Program 5: Rolling a Pair of Dice and Searching a Two-Dimensional Array

Why would someone want to search through an array? The main reason would be to find if a particular value or element exists within the array. This is important for tasks such as checking for the presence of a specific client in a database, verifying an item in an inventory, or finding a specific record and then getting information included in that record.

Let's make searching an array a fun exercise by **simulating** rolling a pair of ordinary six-sided dice and storing the *sums* of each roll in our two-dimensional array. Since our array is 3 rows by 4 columns or 12 elements in total, we will simulate rolling the pair of dice 12 times. Each result will be a random number between 2 and 12, with 7 having the greatest probability of occurring. This is because a sum of 7 can be obtained in the greatest number of ways. These (die1, die2) possibilities are (3, 4), (4, 3), (5, 2), (2, 5), (6, 1), and (1, 6). Any other sum occurs with less frequency. For example, there are only 2 ways to get a sum of 3: (1, 2) and (2, 1).

The goal of our program is to have Ozobot determine the indexes of the locations in which a sum of 7 has occurred. The complete program is shown in Figure 8. The variables *die1* and *die2* are defined to be random integers between 1 and 6, inclusive. The variable *sumOfDice* is then defined as the sum of the two dice. The function call to *Set Element* then stores this sum at row and column index locations *i* and *j*.

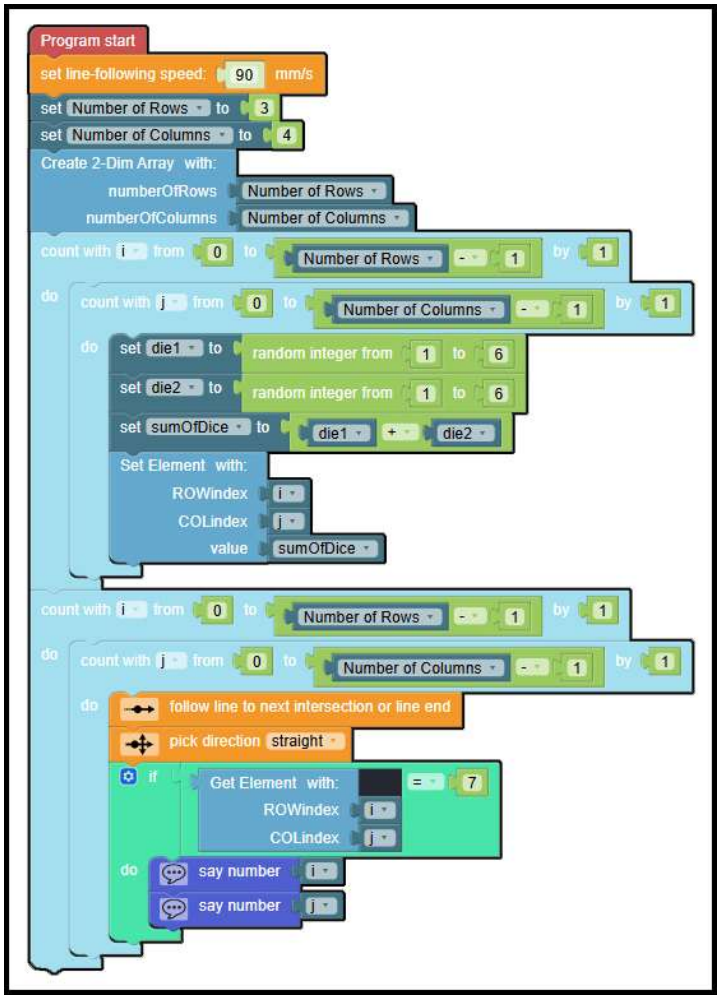


Figure 8

In the final double-nested *count with* loop, the function call to *Get Element* checks to see if the stored value is 7. If so, then Ozobot speaks the value of the indexes of the location. If not, then Ozobot simply moves to the next location on the Ozomap, speaking nothing. In other words, Ozobot has been programmed to stop at locations where the sum of the dice is 7 and then speak the indexes of the location, indicating the results of the search.

It is time to modify your program to that shown in Figure 8! Then run your program in the Ozobot Editor five times, noting in the table below the number of times a sum of 7 is rolled in each run of the program.

Run #	1	2	3	4	5	Average
Number of 7's						

QUESTION 1: What is the theoretical probability, expressed as a number between 0 and 1, with 0 meaning no chance at all and 1 meaning with absolute certainty, for obtaining a sum of 7. *Hint: Complete the following table with the ways in which each sum from 2 through 12 can occur.*

2	3	4	5	6	7	8	9	10	11	12
	(1, 2)				(3, 4)					
	(2, 1)				(4, 3)					
					(5, 2)					
					(2, 5)					
					(6, 1)					
					(1, 6)					

QUESTION 2: With theoretical probabilities in mind, how many of your twelve runs, would you expect, on average, to get a sum of 7? _____

QUESTION 3: How does the average you obtained from running the program compare to what is expected, based on theoretical probability? _____

(This area purposely left blank—lesson continued on the next page)

Program 6: Creating a 3x3 Identity Matrix

An *identity matrix* is a square array for which the elements along the main diagonal (upper left through lower right) are all 1s. The rest of the elements are all 0. An identity matrix is the equivalent of the number 1 in multiplication since the effect of multiplying a given matrix by an identity matrix is to leave the given matrix unchanged. Identity matrices are used in linear algebra as a starting point when solving systems of equations. They are also used when defining the inverse of a matrix.

Your task is to build an OzoBlockly Editor program that creates a 3x3 identity array and then has Ozobot traverse the array, speaking the values of the elements in the array. You will be making use of a larger copy of the Ozomap shown in Figure 9. The only stipulation is that you are allowed to have **just one** "Set Element" block in your program.

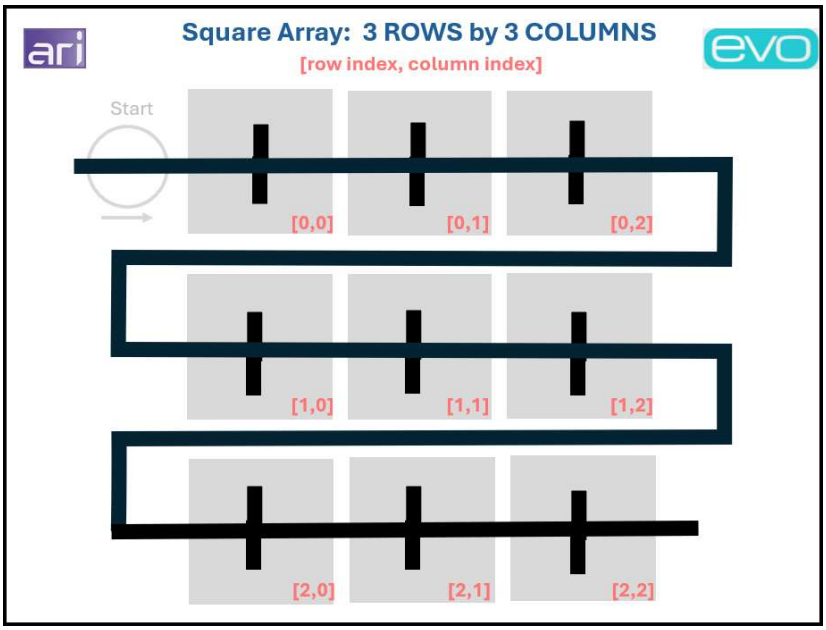


Figure 9

Demonstrate your running program to your teacher.

(This area purposely left blank—lesson continued on the next page)

Program 7: Dealing a 7-Card Poker Hand

Most people enjoy card games! So, in this final exercise, you will study an OzoBlockly Editor program that allows both Ari and Evo to deal random poker hands from a deck of standard playing cards. The trick here is to make sure that any given card in the deck is dealt no more than one time. One way to accomplish this is to map the 52 cards in a standard deck of playing cards to a two-dimensional array, as shown in Figure 10.

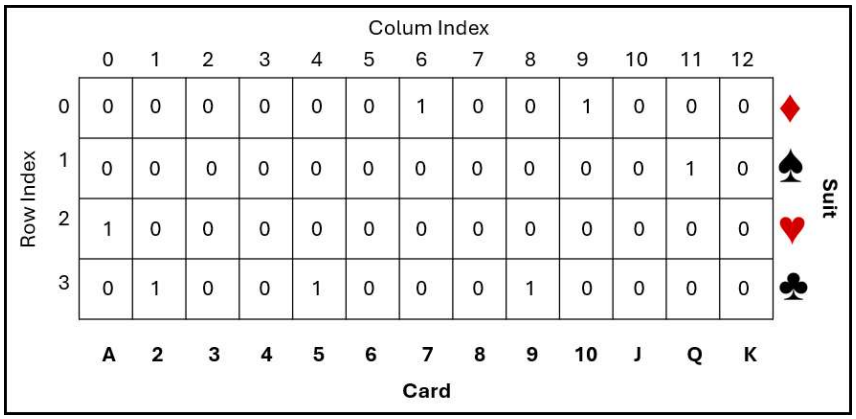


Figure 10

Each row represents a suit—diamonds, spades, hearts, and clubs. The columns represent the cards from Ace through King. The array is initially all zeros. A random number from 0 to 4 represents the suit and a random number from 0 to 12 represents the card in that suit. When a card has been selected for a poker hand, the value 1 is placed in the two-dimensional array for that card, meaning that the card has been assigned and cannot be randomly assigned again. For example, the 1 at (row index, column index) = (1, 11) means that queen of spades has been assigned to the poker hand. The poker hand represented in Figure 10 contains the seven cards: 10 of diamonds, queen of spades, ace of hearts, 2 of spades, 5 of spades, and 9 of spades.

The Ozomap in which Ozobot deals a 7-card poker hand is shown in Figure 11.

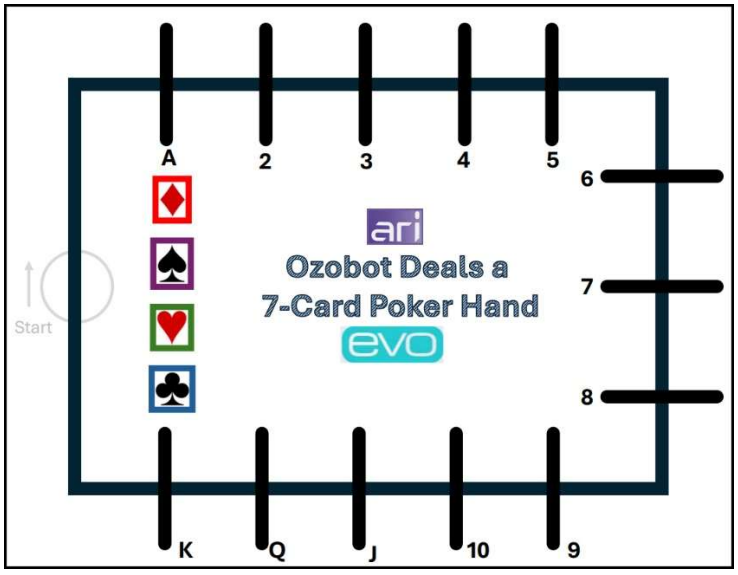


Figure 11

Ozobot begins at the gray circle on the far left facing the direction of the arrow. Ozobot then circles around the large loop four times, once each for diamonds, spades, hearts, and clubs, in that order. Ozobot's top light is set to red while circling for diamonds, purple for spades, green for hearts, and blue for clubs—in agreement with the suit colors shown on the left side of the Ozomap. If a given card has been randomly selected, then Ozobot stops at the card's location and beeps.

A video accompanying this lesson shows Ozobot in action. To keep the video reasonably short, it only shows Ozobot's first loop around the map, circling to indicate any diamond cards that were randomly selected. You will note that the top light turns red and that Ozobot beeps at cards 3, 4, 5, and 6. Wow—nearly a straight flush! But nearly only counts in horseshoes. Though not shown in the video, Ozobot then circles three more times for the remaining suits—spades, hearts, and clubs. In this process three more cards are dealt, which when combined with the four heart cards, gives a poker hand of seven cards.

Now that you have seen how Ozobot traverses the Ozomap while dealing a poker hand, you are ready to investigate its Ozobot Editor program. The program was designed by beginning with the *Two-Dimensional Array Template* program. Figure 12 shows the blocks making up the program with the exception of the function declarations to *create* an array, *set* elements in the array, and *get* elements from the array.

A modular approach was used to design this program, with a main program that makes function calls to procedures with specific tasks. The main program, beginning with the *Program start* block, begins by initializing the line following speed and setting the value of three constants used in the program. Then there are three function calls, one to create the array, one to deal the hand, and one for Ozobot to display the poker hand on the Ozomap. The top light is then turned off to indicate that the program is done.

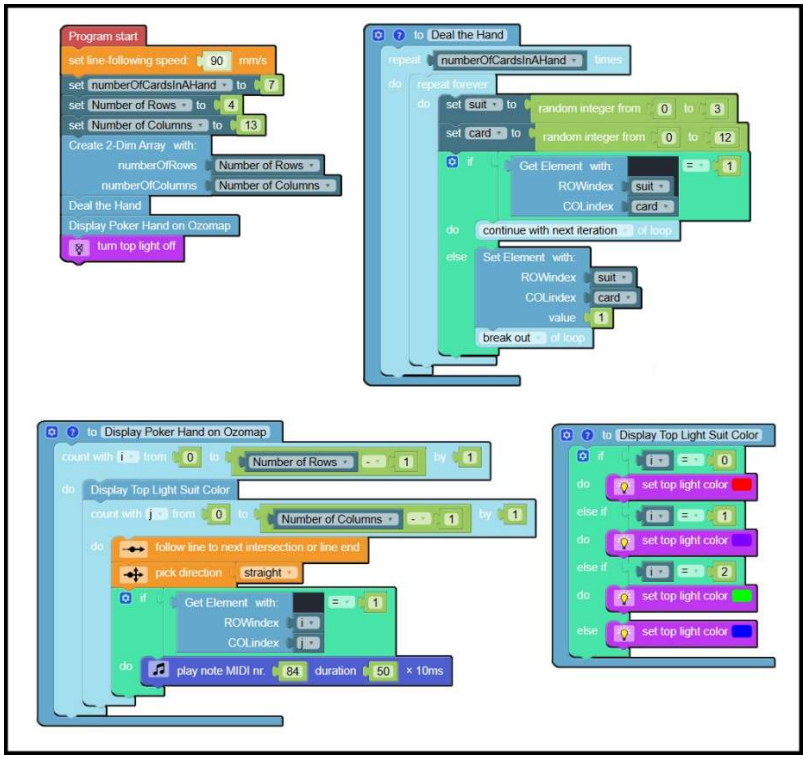


Figure 12

Deal the Hand Function

Keep in mind that the initial value for each element in the array is set to zero by default. Zero means the card has not yet been dealt. A loop is then set up that executes once for each of the seven cards in the poker hand. Nested in that loop is a repeat forever loop. But do not worry, as you can see that the loop can be exited with a *break out of loop* block. Inside the loop, the suit is set to a random integer from 0 to 3, and card to a random integer from 0 to 12, as per the array design discussed in relation to Figure 10. An if block then checks to see if the element (suit, card) has already been dealt, *i.e.*, contains the value 1. If so, the next iteration of the loop forever loop will try again for a random card that has not already been assigned. If not, then the element (suit, card) is still available and the value 1 is assigned to it to indicate that it is now a part of the poker hand. This process is repeated for each of the seven cards in the poker hand.

A few more remarks are in order on the use of the *repeat forever* loop. While the odds for randomly selecting the same element of the array that has already been assigned are small, it can happen. If you were to modify the program to deal a 7-card poker hand for, say, 6 players, then the odds are significantly higher, as you would be dealing a total of 42 cards out of a deck of 52 cards. There would likely by a lot of "collisions" that you would need to account for. The *repeat forever* loop is the solution for this problem, knowing that you can *break out of the loop* when appropriate.

Display Poker Hand on Ozomap Function

This function has Ozobot making four trips around the Ozomap, one trip for each of the suits diamonds, spades, hearts, and clubs. While traveling around the loop, Ozobot stops at those cards of that suit that are part of the poker hand, displaying the color of the suit shown on the Ozomap. The *Display Top Light Suit Color* function is used to set this color. If the card at Ozobot's current location is to be dealt, *i.e.*, has an array value of 1, then a midi note is played for 10 x 50ms = 500ms = 0.5 second.

It's Time to Run the Program

Armed with an understanding of the blocks in this 7-card poker hand simulation, you are now ready to create and run this program with the Ozobot Editor.

Program Modification Challenge

Wait a second. Poker isn't just for one player! Make modifications to your program so that Ozobot will deal a 7-card poker hand to more than one player and do so from the same deck of cards. Test your program with three players. No two players should have the same card. For example, Jack and Jen both cannot have the queen of hearts.

Question 1: What happens to the frequency of execution of the block *"continue with next iteration*" as the number of players increases?

Question 2: What would happen if you tried running the program with eight players?