**Complexity Order**

The complexity of our algorithm can be broken down into the two arrays and the three major functions or loops found within the javascript code: function pointThis, function changeStateArray, and the while loop to run through the program. Note that we have set the 'n' value as 1,000 by declaring that the while loop run until it reaches 1,000 times, but this analysis characterizes the complexity with n as if it is being inputted. There are also other components to the program such as drawing the grid and assigning initial values, but these all contain a complexity of O(1) as they are statements not depending on an increasing / decreasing value.

There are two arrays: one one-dimensional array that collects color data and one two-dimensional array that collects coordinate data (x-axis and y-axis). For the coordinate-collecting array, since it is a two-dimensional array and the matrix for it may not be square at the given time of search, the complexity is O(n+k) where n is the number of rows and k is the number of columns. If a matching location is found, the program goes to the color array at the same numbered index, which does not require a search since it already knows the index making the complexity of this O(1). Therefore, the overall complexities of the arrays is O(n+k).

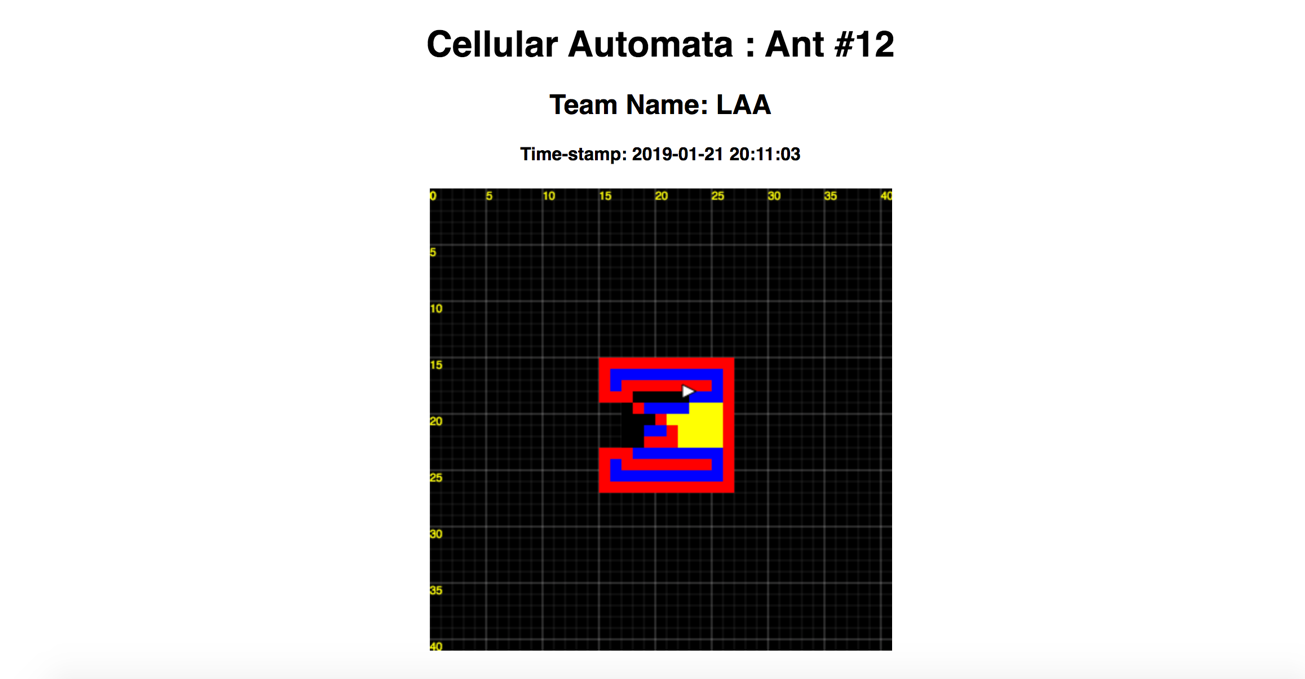
The function pointThis is made of a series of if-else statements that take the direction integer and compare for each statement. This does not require an input n, but merely depends on comparing values and then statements that change more values, but there are no loops or searches and therefore these if-else statements have a complexity of O(1).

The function changeStateArray is made of if-else statements as well as nested ifs. These if-else statements have a complexity of O(1) as they do not depend on an increasing / decreasing input and are not loops or other complex actions. The nested ifs are also non-dependent on an increasing / decreasing input. Therefore, this function has an overall complexity of O(1).

The while loop runs in an incremental fashion and it contains if-else statements. These if-else statements check the value of the current array state, change that assigned value and then more statements to change other values. Since these if-else statements do not depend on an inputted or incremental value, but are just comparisons and changing values, they maintain a complexity of O(1). However, since the while loop continues iterations incrementally, it has a complexity of O(n).

Combining the complexities discussed above for the arrays and functions, there are multiple complexities of O(1) as well as complexities of O(n) and O(n+k). In Big-O analysis, the greatest value is taken as the overall complexity and in this case, the overall complexity is O(n+k).

**Program Output**

****