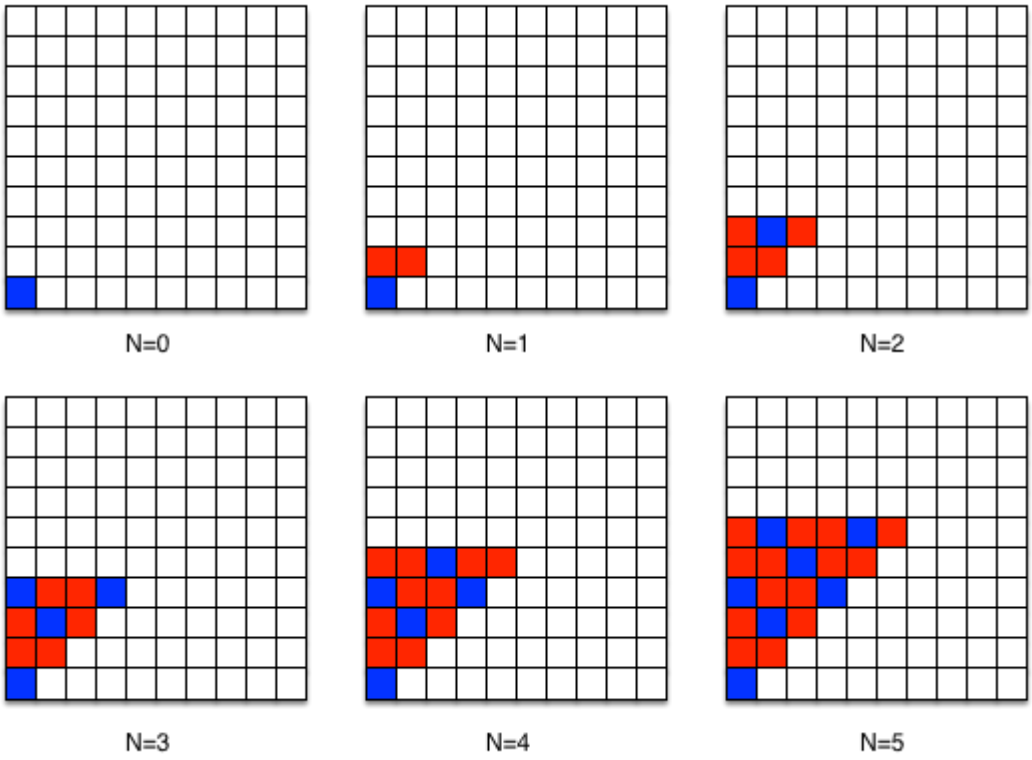
A Pattern of Squares

For our second example, we will look at a pattern of squares drawn on a grid. You may wonder why a programmer would be interested in drawing squares on a grid. Beyond this example serving us well for analyzing patterns in general, computer graphics ultimately boil down to drawing colored pixels on a 2D grid (the screen). In this particular example, we have an algorithm that is parameterized over one integer *N* and produces a pattern of red and blue squares on a grid that starts all white. The output of the algorithm for $N = 0$ to $N = 5$ is as follows:



To devise the algorithm, we should work through steps 1–4 (as we should with all problems). The next video walks through these steps to illustrate how we could come up with this algorithm.

We note that there are *many* correct algorithms for this problem. Even if we restrict ourselves to the ones we can come up with naturally (that is, as a result of working through steps 1–4, rather than trying something bizarre), there are still many choices that are equivalent and correct. Which algorithm you come up with would be determined by how you approach step 1.

The algorithm in the video works from left to right, filling in each column from bottom to top. If we had worked step 1 by working from the top down, filling in each row from left to right, we might have ended up with the following slightly different algorithm instead:

```
Count down from N to 0 (inclusive), call each number you count "y" and
  Count from 0 to y (inclusive), call each number you count "x" and
    if (x + y is multiple of 3)
      then place a blue square at (x,y)
    otherwise place a red square at (x,y)
```

Of course, those are not the only two ways. You could have worked across the rows from the bottom up going right to left, and come up with a slightly different (but also equivalent) algorithm. Or even an entirely different approach, such as filling in the entire "triangle" with red squares, then going back to fill in the blue squares.

We emphasize this point because it is important for you to understand that there is always more than one right answer to a programming problem. You might work a problem and come up with a correct solution but find that it looks completely different from some other solution you know to be correct (*e.g.*, the ones provided for some of the problems in this book, or a teacher's solutions to an exam). Understanding this possibility is important so that you will not incorrectly think that a right answer is wrong because you have seen a different right answer. Not only is that experience frustrating, but it hinders your learning.

Completed