Assignment 1: Implement Elementary Cellular Automata in C# using bitwise operators

Submission

Please submit your answer to CANVAS. I only want to get the code files (ending in *.cs) and if possible put all code into a single file. Please put your name and student number at the top of the file.

Introduction

As discussed in class an elementary cellular automata is sequentially produced by using the information in a current array of 0's and 1's to calculate the 0's and 1's in the next array according to a specific rule set. To update the value at position n in the next array, the triplet of values in the previous array at position n+1, n and n-1 are used. A particular rule informs the program how to update based on each possible triplet configuration. For example:

{1,1,1}->0; {1,1,0}->1; {1,0,1}->1; {1,0,0}->0; {0,1,1}->0; {0,1,0}->1; {0,0,1}->0; {0,0,0}->0;

Requirements

Encapsulate your solution in an object. Each array must be encoded in a uint and the rule must be coded in a single byte. The object must expose methods to

1) Initialise the first array by either a random sequence of bits or single bit switched on.
2) Output the bit representation of an array.
3) Have a run method which results in the full sequence being output to screen.
4) The user should be able to specify the rule and number of steps (taking care to ensure valid values).
5) A function to output the rule (like that shown above) is to be provided.
6) Decompose the effort into sensible methods if possible.

A possible sequence of input/output is shown below:

```
4
This CA in C#.
Please enter the rule: any number between 0 and 255
30
Please enter the number of steps: any number between 0 and 200
50
Please enter the type of initialisation: 0 for random, 1 for a single non zero entry in the middle
1
You entered rule:
(0,0,0)->0
(0,0,1)->1
(0,1,0)->1
(0,1,1)->1
(1,0,0)->1
(1,0,1)->0
(1,1,0)->0
(1,1,1)->0
The initialisation is:
00000000000000010000000000000000
The CA is:
00000000000000010000000000000000
00000000000000111000000000000000
00000000000001100100000000000000
00000000000011011110000000000000
00000000001100100010000000000000
00000000011011110111000000000000
00000000110010001001000000000000
00000001101111001111110000000000
00000011001000111000010000000000
00000110111101100100011100000000
00001100100001011110110010000000
00011011110011010000101111000000
00110011000111001100110100010000
01101111011001110111001101110000
11001000010111000100111001001000
11011110011010010111110011111110
10010001110011111010000111000000
11111011001110000110011001000001
00000010111001001101110111100011
10000110100111111001000100010110
11001100111100000111101110110100
10111011100010001100001000100111
00100010010111011010011101111100
01110111110100010011110001000010
```