KAZAKH BRITISH TECHNICAL UNIVERSITY

REPORT

Game of life

Basics of Computer Simulation

Prepared by: Balganym Tulebayeva
Checked by: Assel Akzhalova

Almaty 2018

The Game of Life was invented in 1970 by the British mathematician **John Horton Conway**. Conway developed an interest in a problem which was made evident in the 1940's by mathematician **John von Neumann**, who aimed to find a hypothetical machine that had the ability to create copies of itself and was successful when he discovered a mathematical model for such a machine with very complicated rules on a rectangular grid. Thus, the Game of Life was Conway's way of simplifying von Neumann's ideas. It is the best-known example of a cellular automaton which is any system in which rules are applied to cells and their neighbors in a regular grid.

I realized Conway's Game of Life by two ways on C#:

1) n x m algorithm (just iterating for all grid)

2) optimized algorithm by using 'Threads'

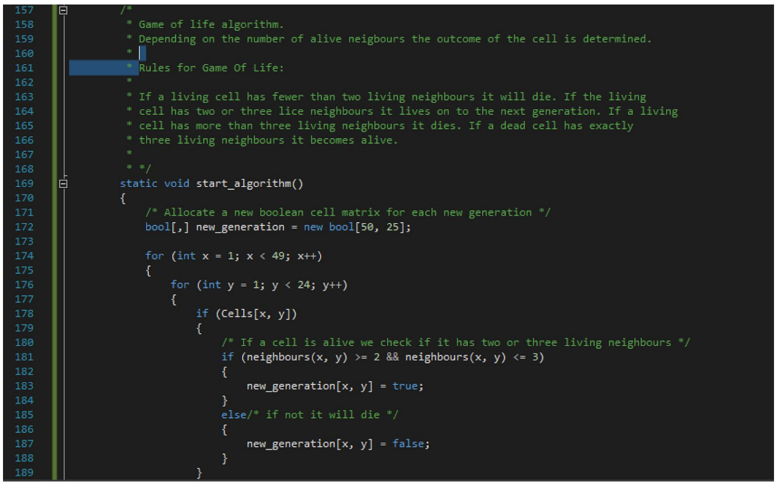
1) In my first realization I have:

- x and y positions of the cursor
- two cell matrices (50 x 25). First is current generation, the second one for next generation. Each cell is a boolean that is either alive or dead (0 or 1);

First my code reads the position of the cursor in the console and starts the game when the key 'S' is pressed.

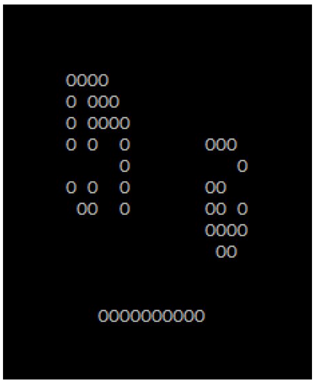
I iterate through my grid and for each cell calculate the number of its alive neighbours. By the information about current state of the cell and the number of its alive neighbours, I update the next_generation matrix;

Here is the screenshots of the game and code:



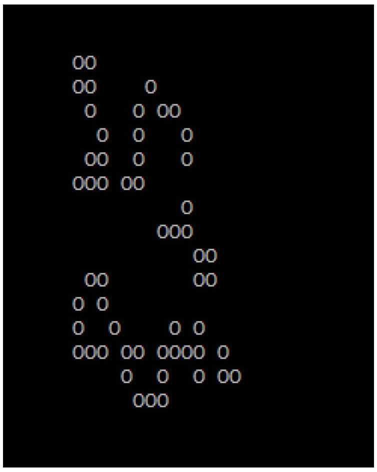
```
        /*
         * Game of life algorithm.
         * Depending on the number of alive neigbours the outcome of the cell is determined.
         * 
         * Rules for Game Of Life:
         *
         * If a living cell has fewer than two living neighbours it will die. If the living
         * cell has two or three lice neighbours it lives on to the next generation. If a living
         * cell has more than three living neighbours it dies. If a dead cell has exactly
         * three living neighbours it becomes alive.
         *
         * */
        static void start_algorithm()
        {
            /* Allocate a new boolean cell matrix for each new generation */
            bool[,] new_generation = new bool[50, 25];

            for (int x = 1; x < 49; x++)
            {
                for (int y = 1; y < 24; y++)
                {
                    if (Cells[x, y])
                    {
                        /* If a cell is alive we check if it has two or three living neighbours */
                        if (neighbours(x, y) >= 2 && neighbours(x, y) <= 3)
                        {
                            new_generation[x, y] = true;
                        }
                        else/* if not it will die */
                        {
                            new_generation[x, y] = false;
                        }
                    }
```

Initial state:

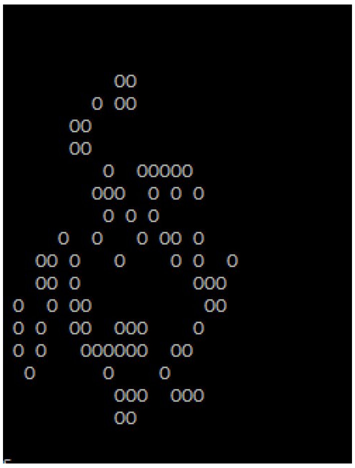


```
OOOO
O OOO
O OOOO
O O  O       OOO
     O          O
O O  O       OO
 OO  O       OO O
             OOOO
              OO


   OOOOOOOOOO
```

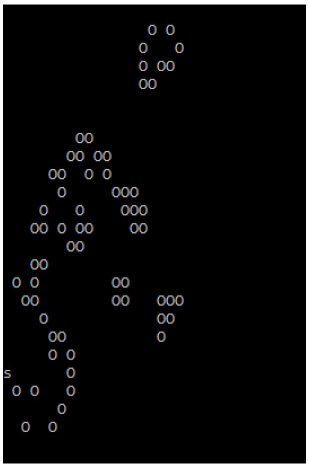
Next generations:



```
OO
OO    O
 O   O OO
  O  O   O
 OO  O   O
OOO OO
         O
       OOO
          OO
 OO       OO
O O
O  O    O O
OOO OO OOOO O
    O  O  O OO
     OOO
```

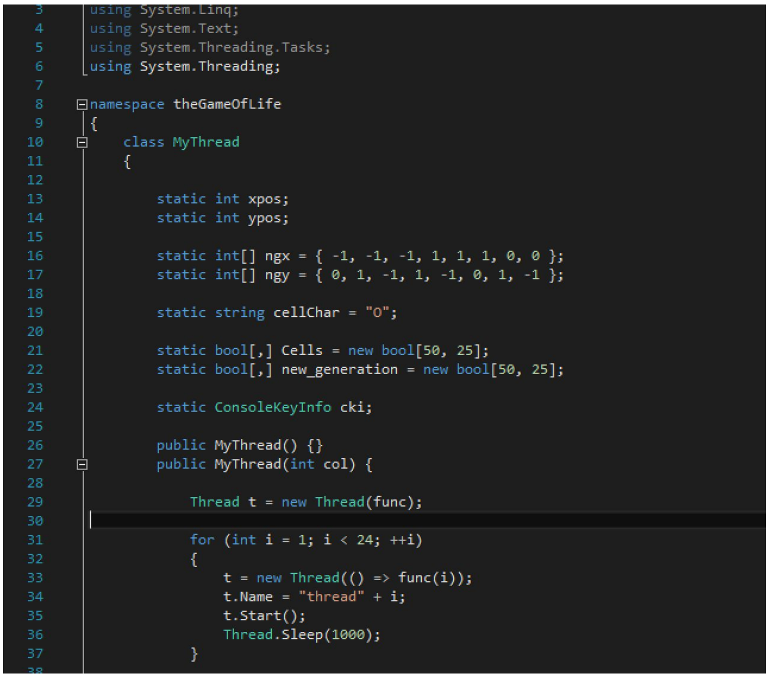


```
         OO
       O OO
     OO
     OO
        O  OOOOO
       OOO  O O O
        O O O
    O  O   O OO O
  OO O   O    O O  O
  OO O          OOO
O  O OO          OO
O O  OO  OOO    O
O O   OOOOOO  OO
 O      O    O
         OOO  OOO
         OO
```



```
                O O
               O   O
               O OO
               OO


        OO
       OO OO
     OO  O O
      O     OOO
    O   O    OOO
   OO O OO    OO
       OO
   OO
 O O        OO
  OO        OO   OOO
    O            OO
     OO          O
     O O
s      O
 O O   O
      O
  O  O
```

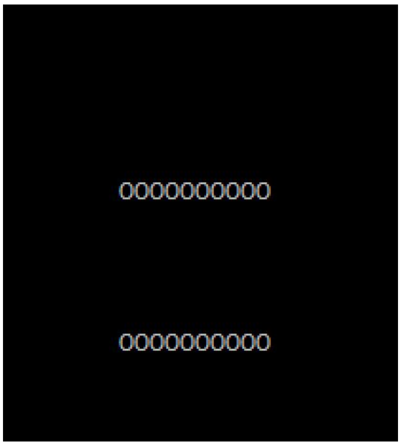
2) The second realization with Threads is the same as previous one. I just add MyThread class. There I created 25 threads (1 x 50), so I just start all of them by iterating, then when all of threads finished the first algorithm, I update my grid.



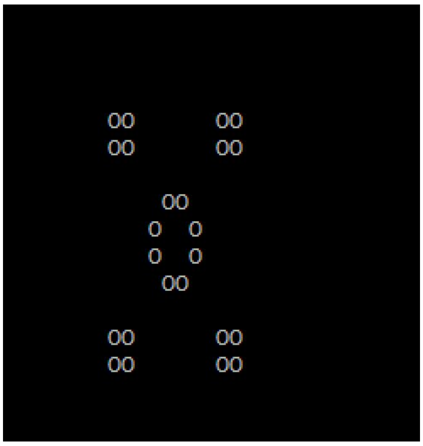
```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace theGameOfLife
{
    class MyThread
    {

        static int xpos;
        static int ypos;

        static int[] ngx = { -1, -1, -1, 1, 1, 1, 0, 0 };
        static int[] ngy = { 0, 1, -1, 1, -1, 0, 1, -1 };

        static string cellChar = "O";

        static bool[,] Cells = new bool[50, 25];
        static bool[,] new_generation = new bool[50, 25];

        static ConsoleKeyInfo cki;

        public MyThread() {}
        public MyThread(int col) {

            Thread t = new Thread(func);

            for (int i = 1; i < 24; ++i)
            {
                t = new Thread(() => func(i));
                t.Name = "thread" + i;
                t.Start();
                Thread.Sleep(1000);
            }
```

Initial state

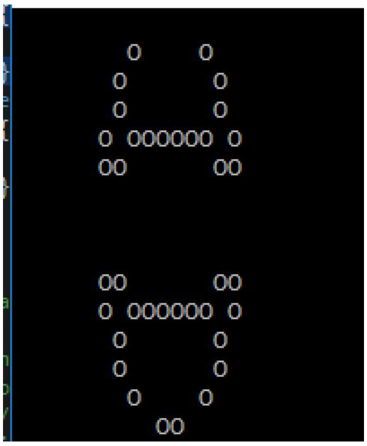
Next generations



```
0000000000




0000000000
```



```
00      00
00      00

    00
   0  0
   0  0
    00

00      00
00      00
```



```
  0    0
 0      0
 0      0
0 000000 0
00      00



00      00
0 000000 0
 0      0
 0      0
  0    0
    00
```