ATTRACTOR TREES OVER GAME OF LIFE RULES (NORMAL AND DIFUSSION)

ALUMNO: BASTIDA PRADO JAIME ARMANDO

PROFESOR: JUÁREZ MARTÍNEZ GENARO

GRUPO: 3CM5

Diciembre 2019

Índice

1. INTRODUCTION

In this practice we tried to do a program in which all of the states, that a Game of Life of 2x2, 3x3, and 4x4 could take, were represented as strings of 0's and 1's, to then be codified as an integer which would represent a concrete state. As the evolution of the Game of Life advances, this numbers would be used to find attractors and then to graph a tree based on this numbers.

1.1. IMPLEMENTATION

To implement this, we treated matrices as numbers and viceversa, so for example, if we had the following Game of Life board:

0	1
1	1

Figura 1: A Game of Life state

We interpret it as a 7. And for example, if we had this:

0	0
1	1

Figura 2: A Game of Life state

We interpret it as a 3.
So, for a 2x2 Game Of Life board, the algorithm iterates over 2^4 different combinations, starting at:

0	0
0	0

Figura 3: A Game of Life state

And ending at:

1	1
1	1

Figura 4: A Game of Life state

It means, from 0 to 15. For a Game Of Life Board of 3x3 the number combinations are 2^9. And so on.

So, the algorithm takes a Game of Life state(one of the combinations), transforms it to a numerical value and stores it in a file in format .csv. The it applies the algorithm for Game of Life and decides whether the algorithm should continue or not, this is based in one of the following cases: the state has been computed already, the game is over (every cell is dead) or the state did not change, meaning it is a "still"life form.

We can see that algorithm here:

```
69          // TRANSFORMAMOS M_OUTPUT EN DECIMAL
70          m_output_val = matrix_to_dec(m_output);
71
72          // ESCRIBÍMOS EL VALOR EN UN .CSV
73          save_writer << (m_output_val + 1) << ",";
74
75          // APLICAMOS ALGORITMO LIFE
76          game_ov_life(m_output, m_state);
77
78          // ESTA BANDERA DECIDE SI SE SIGUEN EJECUTANDO LOS PASOS DE LIFE
79          continue_life = true;
80
81          // TRANSFORMAMOS M_STATE A VALOR DECIMAL
82          m_state_val = matrix_to_dec(m_state);
83
84          // SI YA HABÍAMOS OBTENIDO ESE VALOR DEJAMOS DE ITERAR
85          if(values[m_state_val])
86              continue_life = false;
```

Figura 5: The Algorithm

```
88          // MARCAMOS EL VALOR COMO YA CALCULADO
89          values[m_state_val] = true;
90
91          // HACEMOS LA BUSQUEDA DE POSIBLES CONDICIONES DE PARO
92          if(continue_life)
93          {
94              // BUSCANDO SI YA ESTA MUERTO EL SISTEMA ES DECIR HAY PUROS CEROS
95              if(m_state_val == 0)
96                  continue_life = false;
97
98              // SI NO HA MUERTO EL SISTEMA CHECAMOS SI YA SE REPITIÓ EL ESTADO
99              if(continue_life)
100             {
101                 if(m_output_val == m_state_val)
102                     continue_life = false;
103             }
104         }
105
106         // SI YA SE ACABA EL SISTEMA GUARDAMOS EL ÚLTIMO DATO
107         if(!continue_life)
108             save_writer << (m_state_val + 1);
```

Figura 6: The Algorithm

2. RESULTS FOR RULE B3/S23

The results are stored in a .csv format to be later graphed in Matlab, For example: "11, 511, 0"which means that a state of Life started at "11"then evolved to "511.ªnd then died "0". Note: To be able to work with graphs on Matlab, we cannot have a "0"value so we added 1 to every one of the values, resulting as an example in: "12, 512, 1ïnstead of "11, 511, 0". A sample when running the program using a 2x2 Game Of Life board:

```
1,1
2,1
3,1
4,4
5,1
6,6
7,1
8,1
9,1
10,1
11,11
12,1
13,13
14,1
15,1
16,1
```

Figura 7: Nodes for the 2x2 Game Of Life Attractor Tree

2.1. Attractor tree for the 2x2 Game of Life

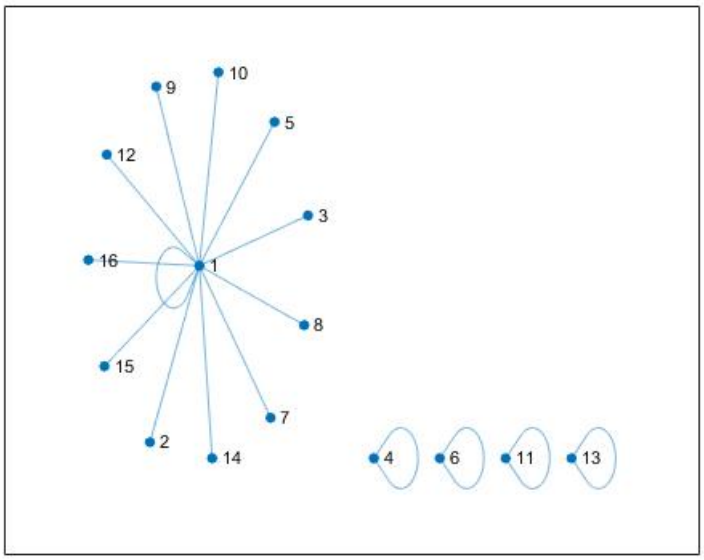
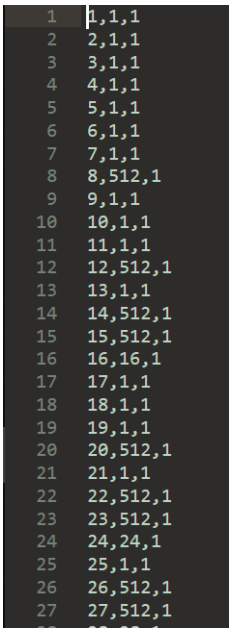


Figura 8: 2x2 Game Of Life Attractor Tree

2.2. Attractor tree for the 3x3 Game of Life

The nodes:



```
1,1,1
2,1,1
3,1,1
4,1,1
5,1,1
6,1,1
7,1,1
8,512,1
9,1,1
10,1,1
11,1,1
12,512,1
13,1,1
14,512,1
15,512,1
16,16,1
17,1,1
18,1,1
19,1,1
20,512,1
21,1,1
22,512,1
23,512,1
24,24,1
25,1,1
26,512,1
27,512,1
```

Figura 9: Nodes for the 3x3 Game Of Life Attractor Tree

The tree:

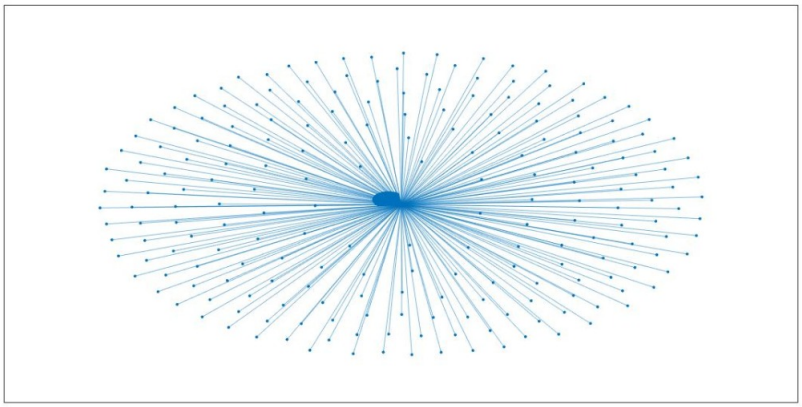


Figura 10: 3x3 Game Of Life Attractor Tree

2.3. Attractor tree for the 4x4 Game of Life

The nodes:

```
 7  7,1
 8  8,8227,8,8227
 9  9,1
10  10,1
11  11,1
12  12,4114,12,4114
13  13,1
14  14,32905,14,32905
15  15,16453,15,16453
16  16,61696,1
17  17,1
18  18,1
19  19,1
20  20,52,52
21  21,1
22  22,171,1
23  23,35,1
24  24,8348,25067,57769,57547,41939,8387,1092,8387
25  25,1
26  26,154,154
27  27,18,1
28  28,4284,47777,43531,40961,1
29  29,137,1
30  30,32828,49643,57763,57451,43385,32873,1098,32873
31  31,16640,20737,43521,1
32  32,61536,81,1
33  33,1
34  34,1
35  35,1
36  36,52
37  37,1
```

Figura 11: Nodes for the 3x3 Game Of Life Attractor Tree

Unfortunately, we couldn't graph the tree for the 4x4 tree. But we could manage to obtain a large list of nodes for the 5x5 tree.

```
620  620,1735,3472,6297883,15733628,8682497
621  621,2048,32519169,32505888
622  622,2030,13314,577,1
623  623,4196223,6294529,163841
624  624,6293261,14762899,10868371,10752659,26480883,14037409,31458241,12988801,13191553
625  625,1585,17970,17746,17746
626  626,1619
627  627,1619
628  628,1050195
629  629,1915,3403,7745,72897,230561,2217025,3821665
630  630,1881,17822,25190551,30417656,4357121,589793
631  631,2015,12614675,12647437,31521837,19935266,20939329,393235,12582913,1
632  632,3147677,32628961,29385697,8521953,132327,6296742,6423470,219298,2589603,230395
633  633,1849,1322,16481,1057
634  634,16778571,634,16778571
635  635,2015
636  636,17827279,33069857,14687201,4473633,263993,25175338,25428462,813346,17727985,820215,6816273,9437194,1
637  637,8390399,25183233,294913,1
638  638,25167053,29886931,21671251,21111123,7480115,14037409
639  639,12584543,18875987,34399,32525919,18433,1
640  640,32506893,20316162,33521684,31777,1047969,32505889
```

Figura 12: Nodes for the 5x5 Game Of Life Attractor Tree

3. RESULTS FOR RULE B2/S7

We proceeded in the same way for the Diffusion rule, and obtained the following results:

3.1. Attractor tree for the 2x2 Game of Life

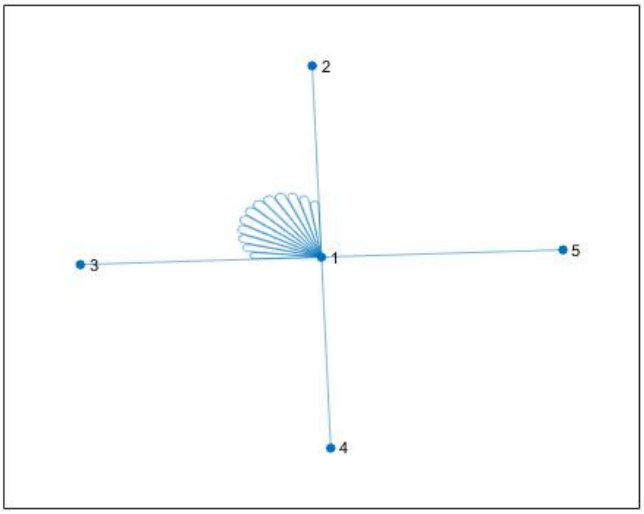


Figura 13: Nodes for the 2x2 Game Of Life Attractor Tree

3.2. Attractor tree for the 3x3 Game of Life

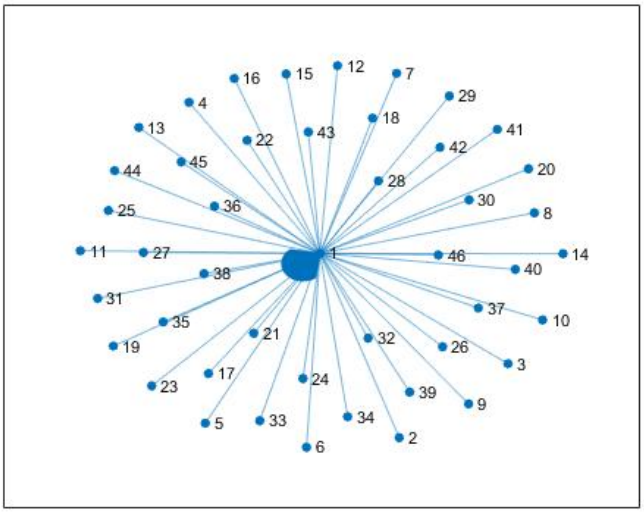


Figura 14: Nodes for the 3x3 Game Of Life Attractor Tree

4. CODE

```
#include <iostream>
#include <cstring>
#include <cmath>
#include <fstream>

using namespace std;

int screenWidth = 5;
int screenHeight = 5;

void fill_array(int *array1, int *array2);
void print_matrix(int *array);
long int matrix_to_dec(int *array);
void game_ov_life(int *m_output, int *m_state);

int main()
{
        int *m_output;
        int *m_state;
        long int i, j;
        long int combination, index;
        unsigned long int m_state_val, m_output_val;
        bool first_time, continue_life;
        long int n_values = pow(2, screenWidth * screenHeight);
        bool values[n_values];
        ofstream save_writer;

        m_output = new int[screenWidth * screenHeight];
        m_state = new int[screenWidth * screenHeight];

        memset(values, false, n_values * sizeof(bool));
        memset(m_output, 0, screenWidth * screenHeight * sizeof(int));
        memset(m_state, 0, screenWidth * screenHeight * sizeof(int));

        save_writer.open("data.dat", ios_base::out | ios_base::trunc);
        if(!save_writer)
        {
                cout << "Error: Cannot open 'strings' file" << endl;
                exit(EXIT_FAILURE);
        }

        // GENERAMOS CSV
        for(combination = 0; combination < pow(2, screenWidth * screenHeight); combination++)
        {
                first_time = true;
                continue_life = true;
                while(continue_life)
                {
                        // ARRRANCAMOS LIFE CON LA CADENA CORRESPONDIENTE
                        if(first_time)
                        {
                                index = 0;
                                for(i = (long int) pow(2, (screenWidth * screenHeight) - 1); i > 0;
                                     i = i >> 1)
                                {
                                        if(i & combination)
                                                m_state[index++] = 1;
                                        else
                                                m_state[index++] = 0;
                                }
                                first_time = false;
                        }

                        // VACIAMOS M_STATE EN M_OUTPUT
                        fill_array(m_state, m_output);

                        // TRANSFORMAMOS M_OUTPUT EN DECIMAL
                        m_output_val = matrix_to_dec(m_output);
```

```
                        // ESCRIBIMOS EL VALOR EN UN .CSV
                        save_writer << (m_output_val + 1) << ",";

                        // APLICAMOS ALGORITMO LIFE
                        game_ov_life(m_output, m_state);

                        // ESTA BANDERA DECIDE SI SE SIGUEN EJECUTANDO LOS PASOS DE LIFE
                        continue_life = true;

                        // TRANSFORMAMOS M_STATE A VALOR DECIMAL
                        m_state_val = matrix_to_dec(m_state);

                        // SI YA HABAMOS OBTENIDO ESE VALOR DEJAMOS DE ITERAR
                        if(values[m_state_val])
                                continue_life = false;

                        // MARCAMOS EL VALOR COMO YA CALCULADO
                        values[m_state_val] = true;

                        // HACEMOS LA BUSQUEDA DE POSIBLES CONDICIONES DE PARO
                        if(continue_life)
                        {
                                // BUSCANDO SI YA ESTA MUERTO EL SISTEMA ES DECIR HAY PUROS CEROS
                                if(m_state_val == 0)
                                        continue_life = false;

                                // SI NO HA MUERTO EL SISTEMA CHECAMOS SI YA SE REPITI EL ESTADO
                                if(continue_life)
                                {
                                        if(m_output_val == m_state_val)
                                                continue_life = false;
                                }
                        }

                        // SI YA SE ACABA EL SISTEMA GUARDAMOS EL  LTIMO  DATO
                        if(!continue_life)
                                save_writer << (m_state_val + 1);
                }
                save_writer << endl;
        }

        save_writer.close();

        return 0;
}

long int matrix_to_dec(int *array)
{
        long int dec_value = 0;
        int exponent = (screenWidth * screenHeight) - 1;
        for(long int i = 0; i < screenWidth * screenHeight; i++)
        {
                if(array[i])
                        dec_value += pow(2, exponent);

                exponent--;
        }

        return dec_value;
}

void game_ov_life(int *m_output, int *m_state)
{
        int nNeighbours;

        auto cell = [&](int x, int y)
        {
                return m_output[y * screenWidth + x];
        };

        for(int x = 0; x < screenWidth; x++)
        {
```

```
                for(int y = 0; y < screenHeight; y++)
                {
                        nNeighbours = 0;
                        for(int i = -1; i < 2; i++)
                                for(int j = -1; j < 2; j++)
                                        nNeighbours += cell((x + i + screenWidth) %screenWidth, (y
                                            + j + screenHeight) %screenHeight);

                        nNeighbours -= cell(x, y);

                        if(cell(x, y) == 1)
                                m_state[y * screenWidth + x] = nNeighbours == 2 || nNeighbours ==
                                    3;
                        else
                                m_state[y * screenWidth + x] = nNeighbours == 3;
                }
        }
}

void print_matrix(int *array)
{
        printf("MATRIX\n");
        for(int y = 0; y < screenWidth; y++)
        {
                for(int x = 0; x < screenHeight; x++)
                        printf("%d ", array[y * screenWidth + x]);
                printf("\n");
        }
}

void fill_array(int *array1, int *array2)
{
        for (int i = 0; i < screenWidth * screenHeight; i++)
                array2[i] = array1[i];
}
```