CONWAY'S GAME OF LIFE

ALUMNO: BASTIDA PRADO JAIME ARMANDO

PROFESOR: JUÁREZ MARTÍNEZ GENARO

GRUPO: 3CM5

Octubre 2019

Índice

1. INTRODUCTION

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.

The game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced players, by creating patterns with particular properties.

1.1. Rules

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead, (or populated and unpopulated, respectively). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbours dies, as if by underpopulation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

2. PROGRAM FUNCTIONALITY

When running the program the user can select in between different simulation options, being the first one the "Life-Like" rule he wants

Figura 1: First option

Then, the user can select if he wants to upload a previous configuration or let the program generate a random number of 1's and 0s, according to the user. A configuration file looks just as a grid of 1's and 0's

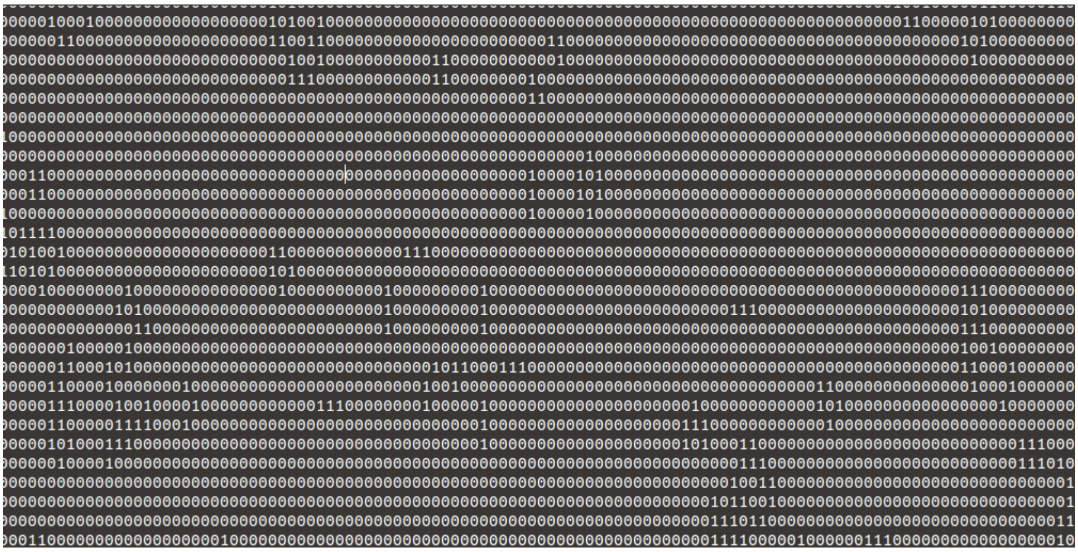

Figura 2: Second option

If the user does not want to upload a configuration, he can choose to generate the cells randomly, indicating the program the percentage of live cells he wants. The percentage can be indicated as: 50 %, 50.5 % or 0.1 %

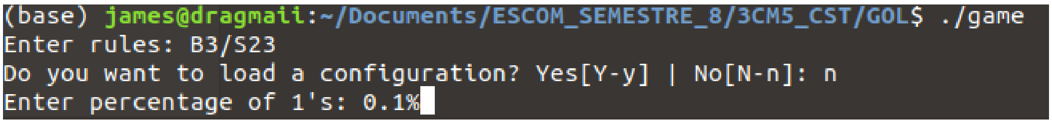

Figura 3: Second option alt

Finally, the user can select between three types of pattern recognition. The first one is [Osc-Still-Chaos] it means you want the program to recognize oscilators, still and chaos(moving) cells and it will paint them diferent colours. The second one is [Static-NonStatic], this tells the program to recognize only among live forms moving and being constant. The last one is [NoRecognition], this tells the program to do not pattern recognition at all.

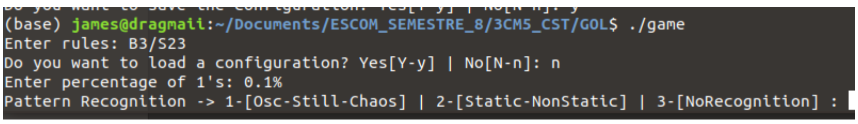

Figura 4: Fourth option

While running the program the user is able to slow down the speed of the simulation by pressing the following buttons: F1(100ms), F2(200ms), F3(300ms), F5(500ms), F10(1s) and F11(2s) and Esc to exit the program.

Before exiting the program it will ask if you want to save the configuration, if so, you can use it to the next time you run the program and continue the simulation

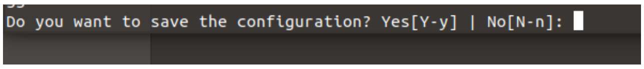

Figura 5: Five option

3. TESTING RULES

3.1. Rule B3/S23 at 50 %

Testing rule B3/S23 with pattern recognition [Osc-Still-Chaos] activated, up to 10,000 iterations, 50 % live cells at the beginning.

3.1.1. 1st Run

System stabilized little after 5,000 iterations. Variance was 2,383,896.6

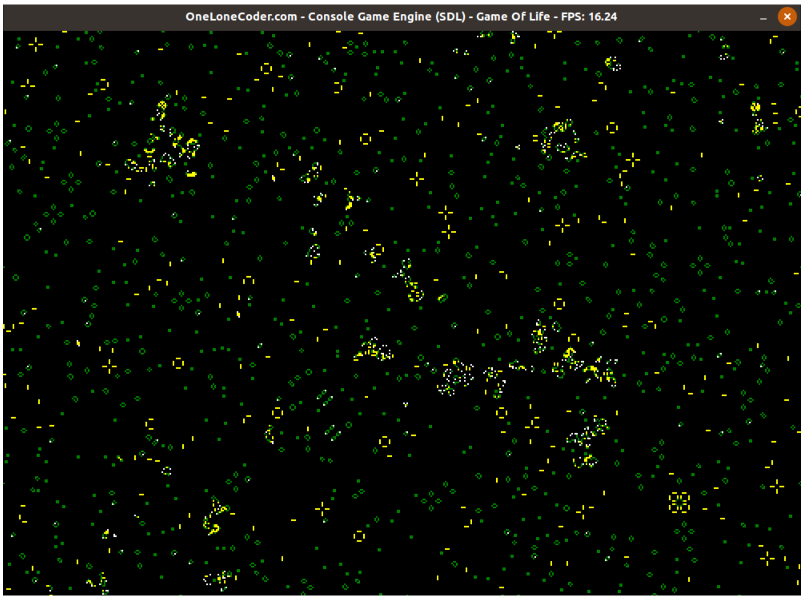

Figura 6: Simulation running

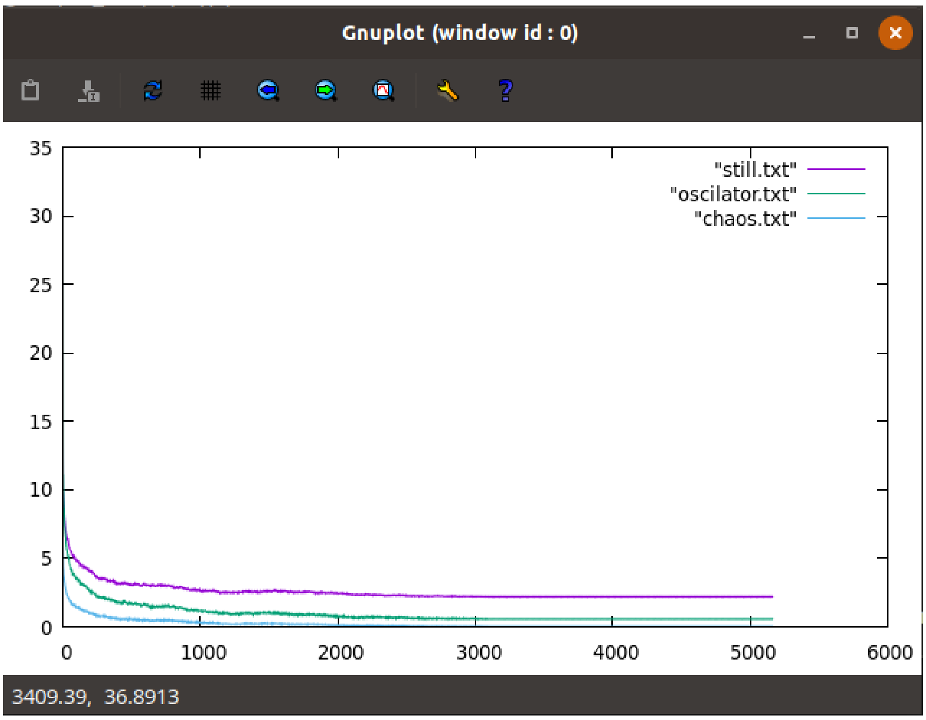

Figura 7: Population Patterns

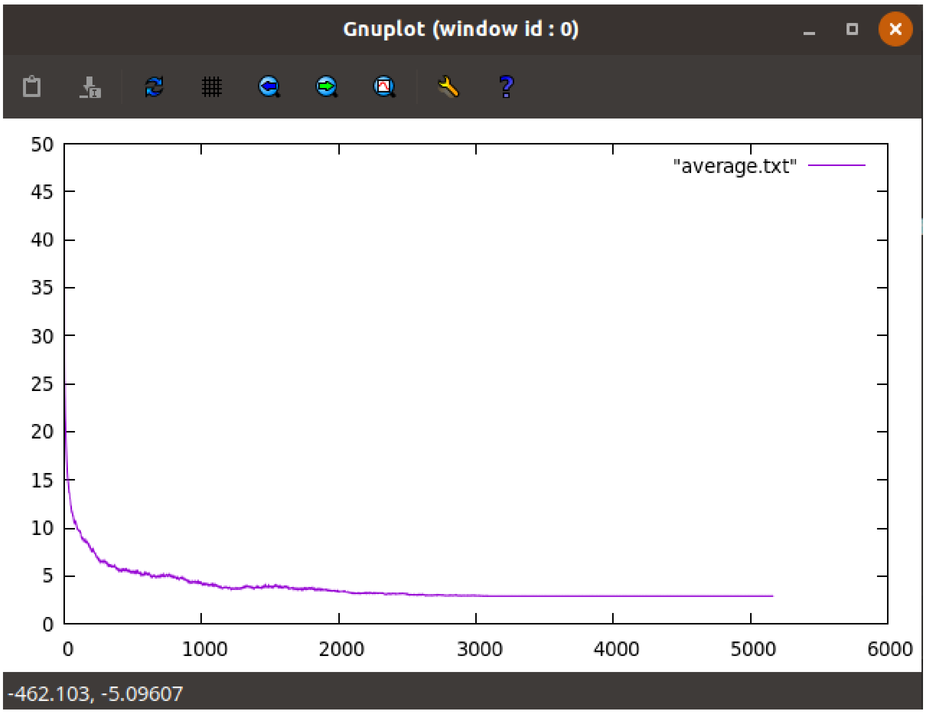

Figura 8: Average live cells

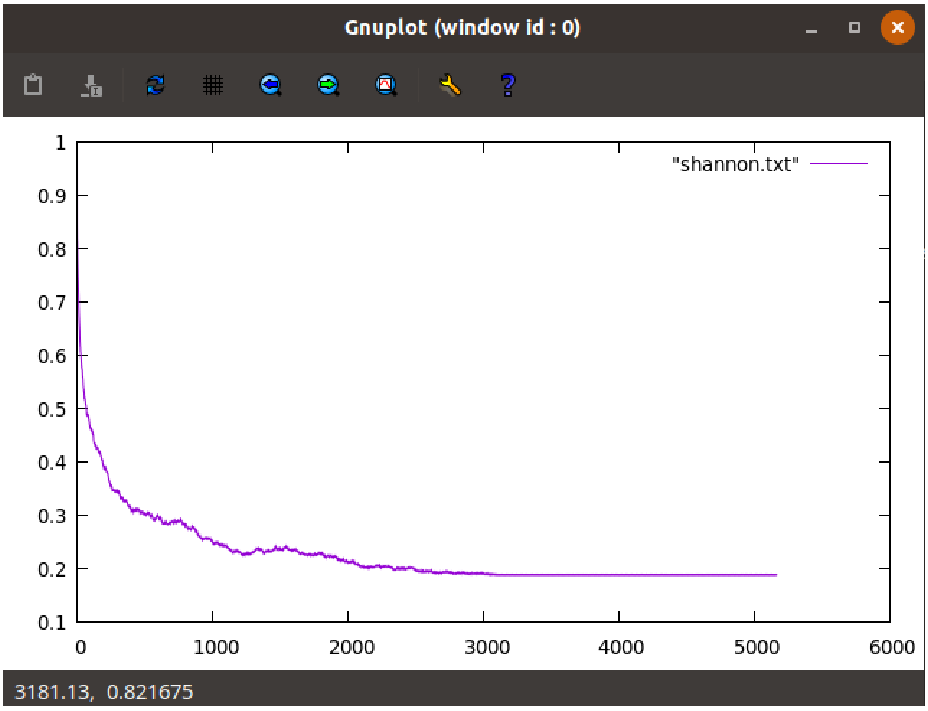

Figura 9: Shannon's Entropy

3.1.2. 2nd Run

System stabilized little after 7,000 iterations. Variance was 3,043,816.4

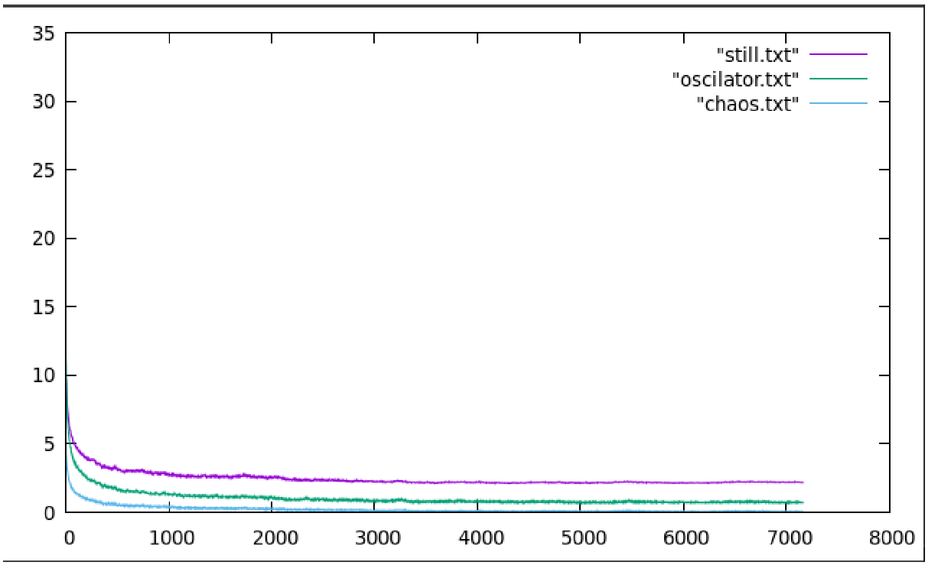

Figura 10: Population Patterns

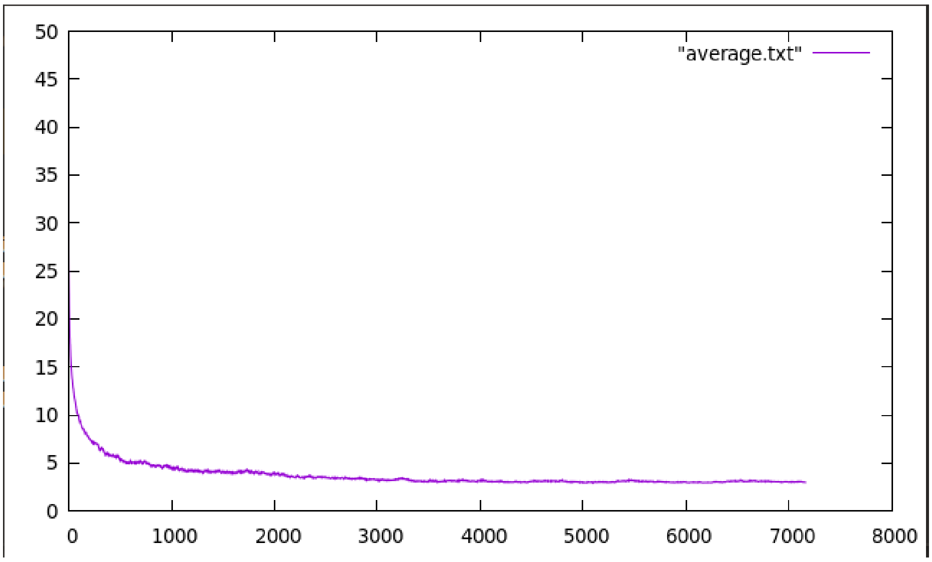

Figura 11: Average live cells

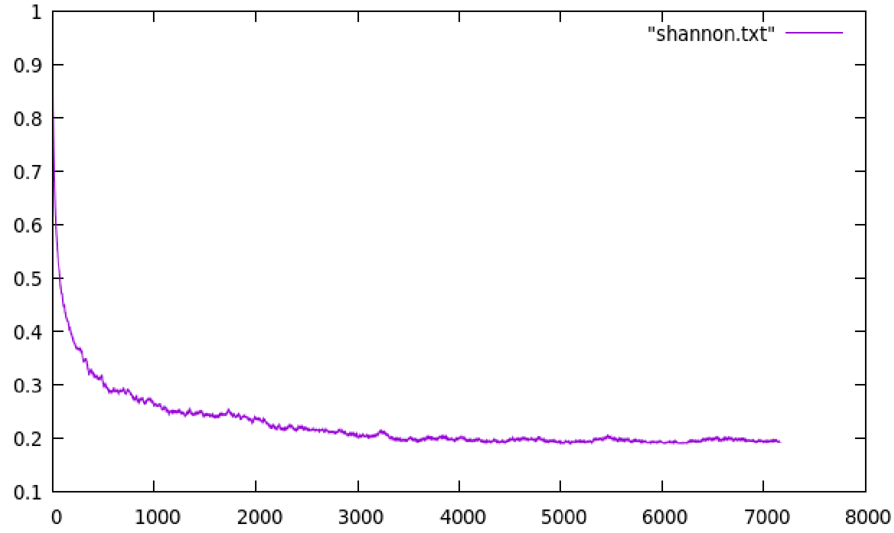

Figura 12: Shannon's Entropy

3.1.3. 3rd Run

System stabilized little after 8,000 iterations. Variance was 3,187,221.7

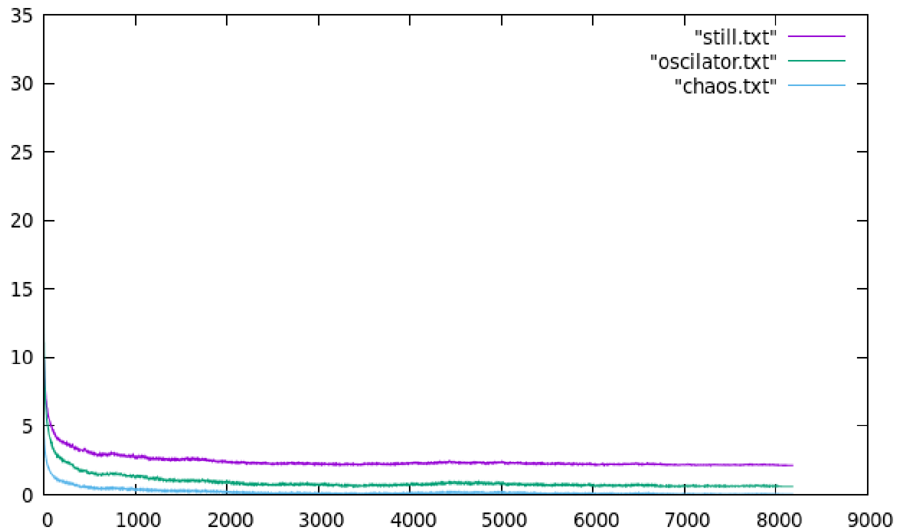

Figura 13: Population Patterns

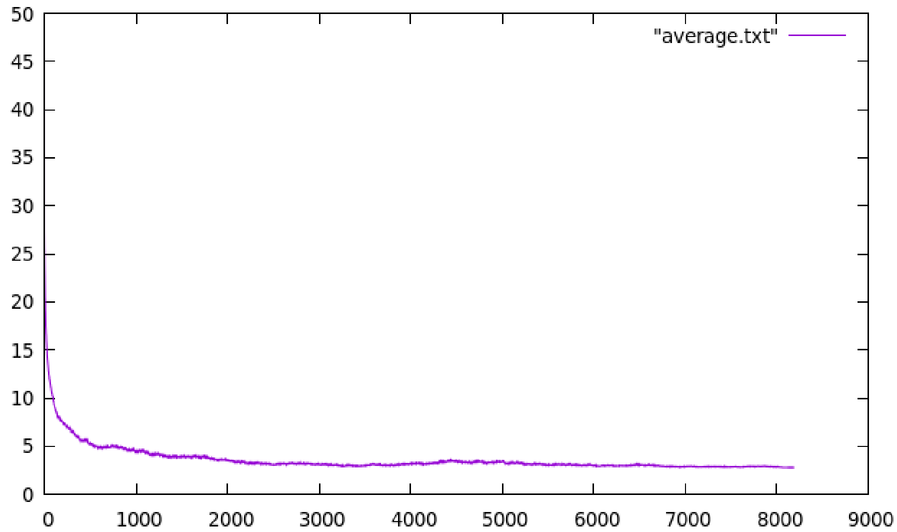

Figura 14: Average live cells

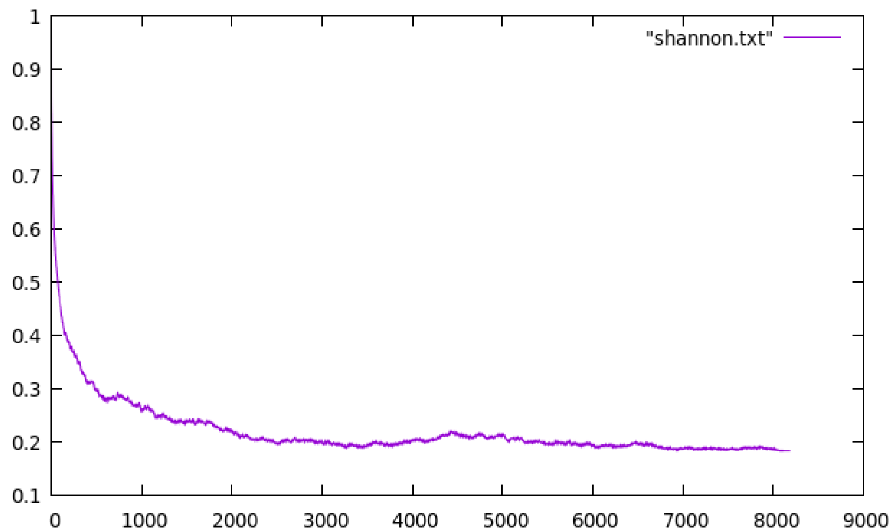

Figura 15: Shannon's Entropy

3.2. Observations: Rule B3/S23 at 50 %

From the three runs we can observe:

- Patterns: The amount of still lives decreases considerably right from the beginning to a percentage below 5 % and stabilizes. The amount of oscillators drops and stabilizes even below than the still lives percentage. The amount of chaotic cells is minimum.
- The average of live cells immediately drops from 50 % to a value near 5 % and stabilizes before reaching 3000 iterations.
- The Shannon's Entropy drops near 0.2 and stabilizes at 5000 iterations.

3.3. Rule B3/S23 at 37 %

Testing rule B3/S23 with pattern recognition [Osc-Still-Chaos] activated, up to 10,000 iterations, 37 % live cells at the beginning.

3.3.1. 1st Run

System stabilized little after 6,000 iterations. Variance was 2,906,110

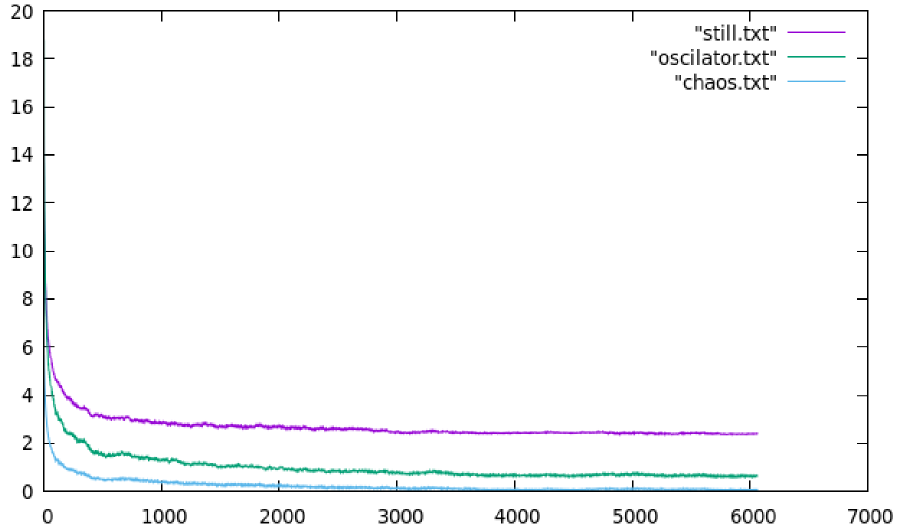

Figura 16: Population Patterns

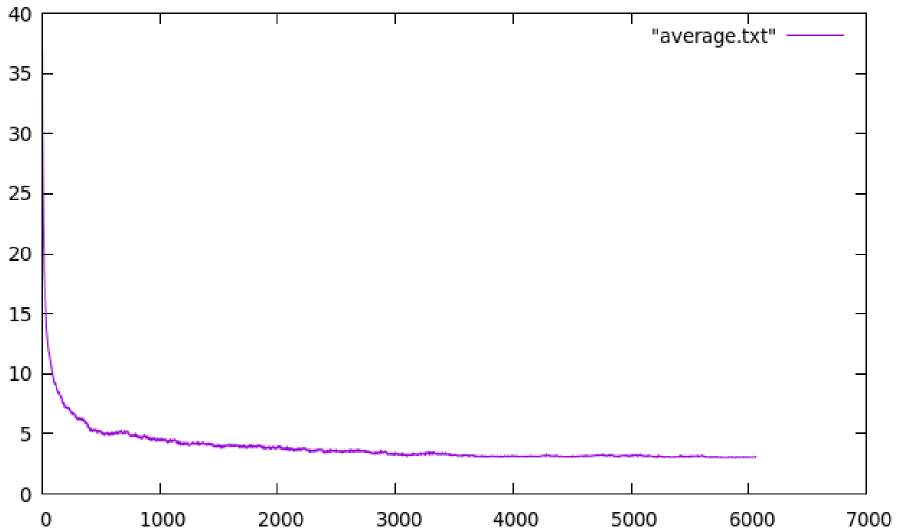

Figura 17: Average live cells

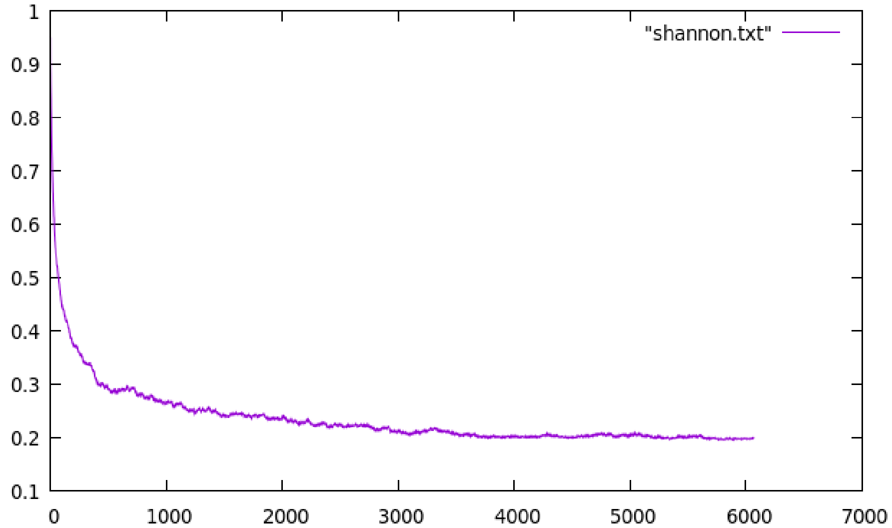

Figura 18: Shannon's Entropy

3.3.2. 2nd Run

System stabilized little before 5,000 iterations. Variance was 2,608,815.1

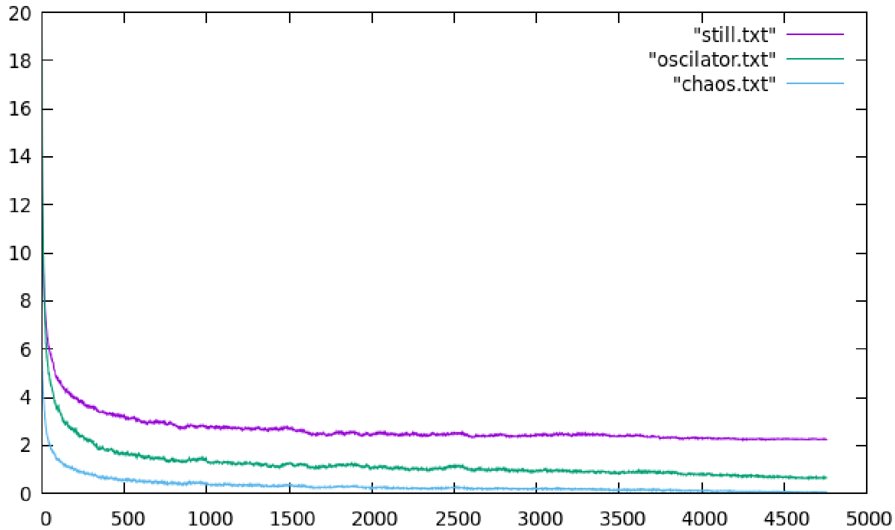

Figura 19: Population Patterns

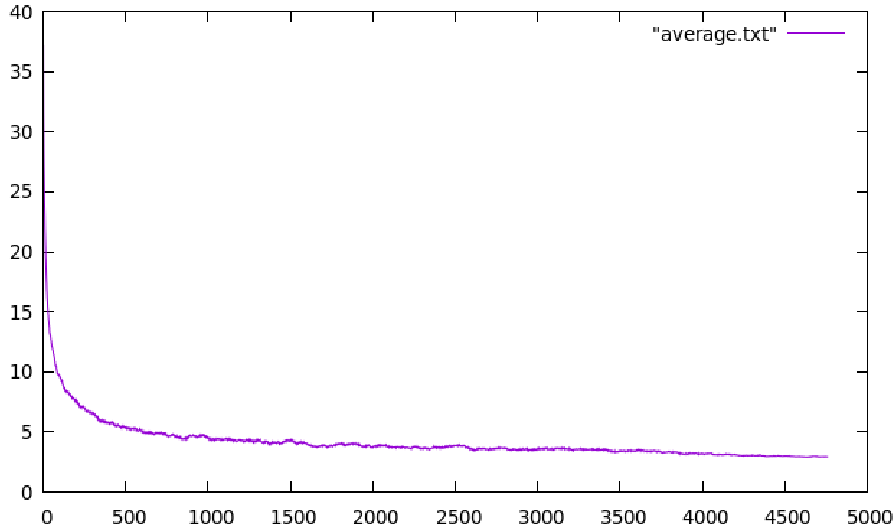

Figura 20: Average live cells

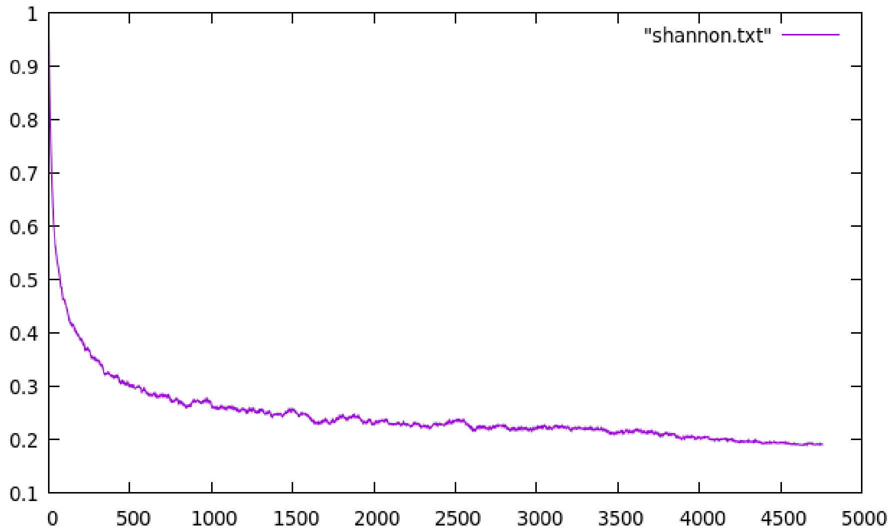

Figura 21: Shannon's Entropy

3.3.3. 3rd Run

System stabilized little after 7,000 iterations. Variance was 3,036,380.6

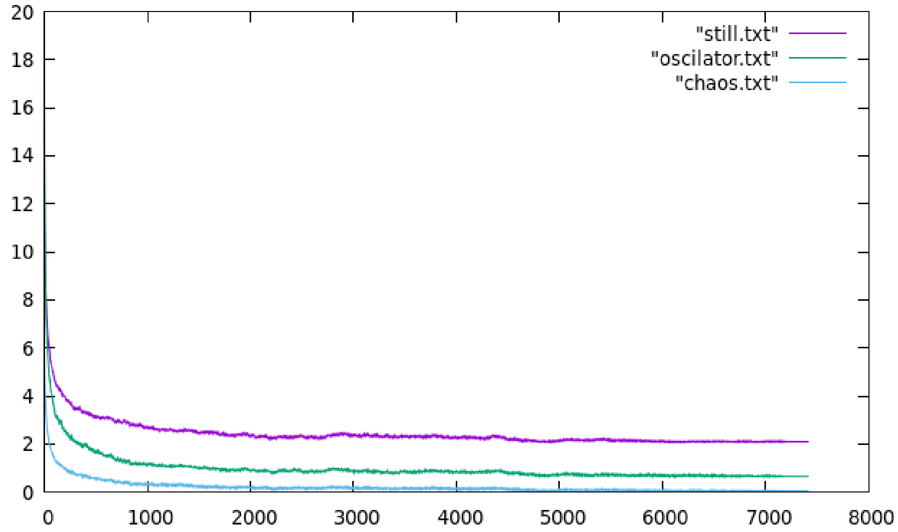

Figura 22: Population Patterns

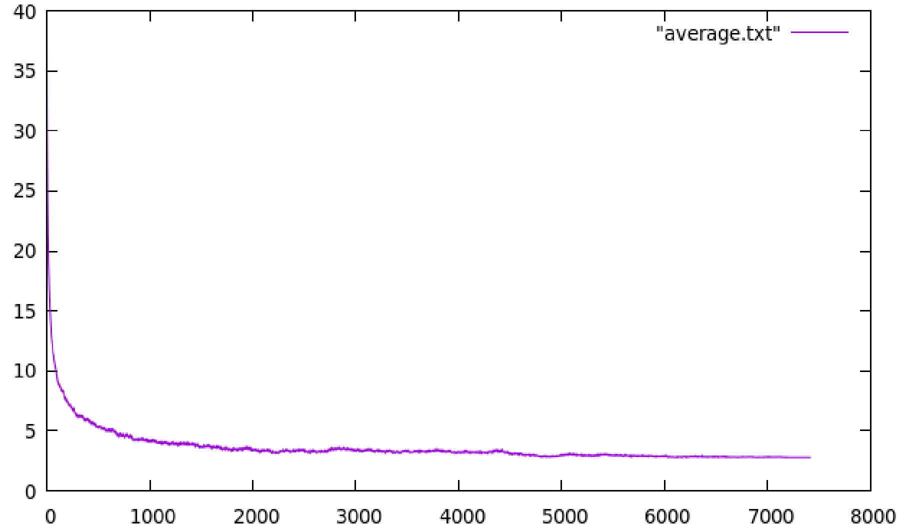

Figura 23: Average live cells

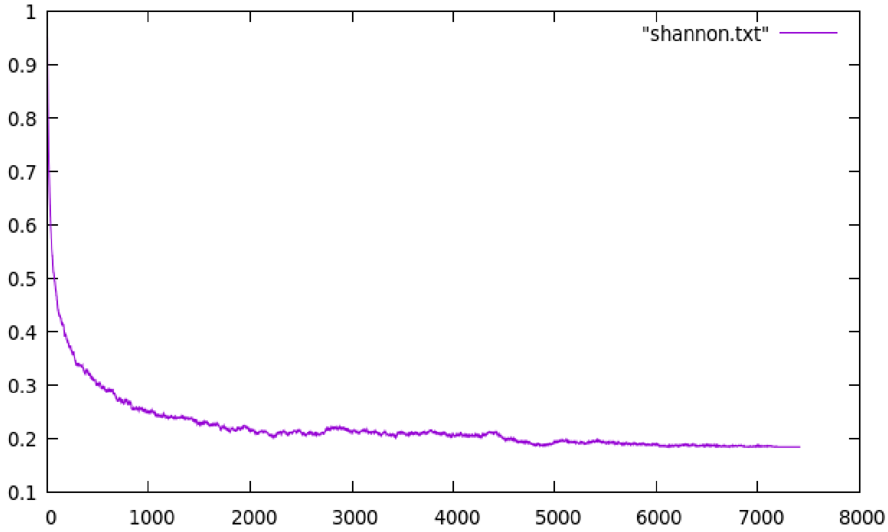

Figura 24: Shannon's Entropy

3.4. Observations: Rule B3/S23 at 37 %

From the three runs we can observe:

- Patterns: The analysis is the same as the one before, just that this time the iterations it takes to stabilize is reduced.
- The analysis is the same as before, but this time the number of iterations it takes to stabilize is lesser.
- It stabilizes quickly than with 50 % live cells.

3.5. Rule B2/S7 at 50 %

Testing rule B2/S7 with pattern recognition [Osc-Still-Chaos] activated, up to 10,000 iterations, 50 % live cells at the beginning.

3.5.1. 1st Run

System never stabilized. Variance was 9,327,339.5

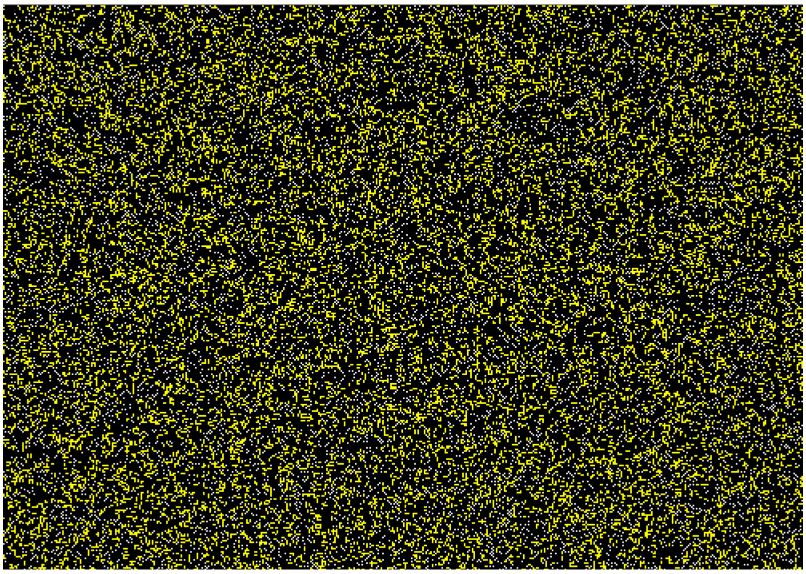

Figura 25: Simulation running

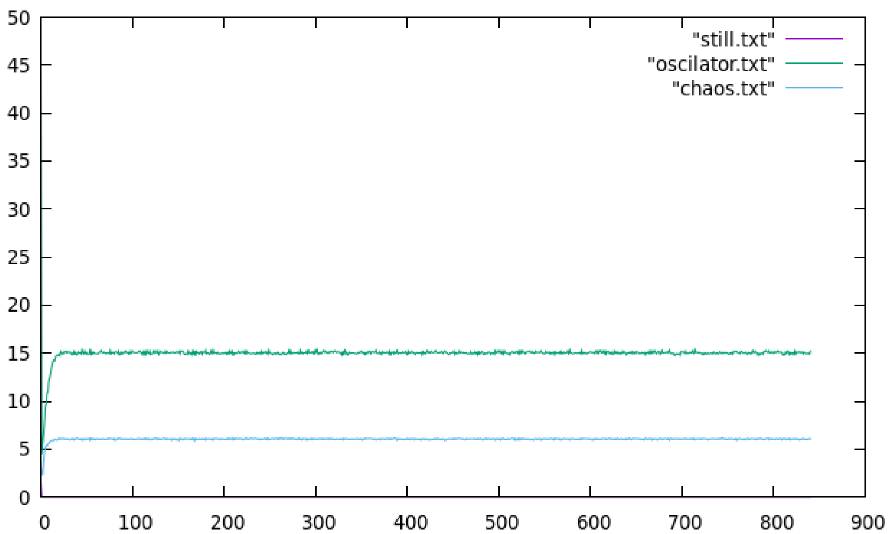

Figura 26: Population Patterns

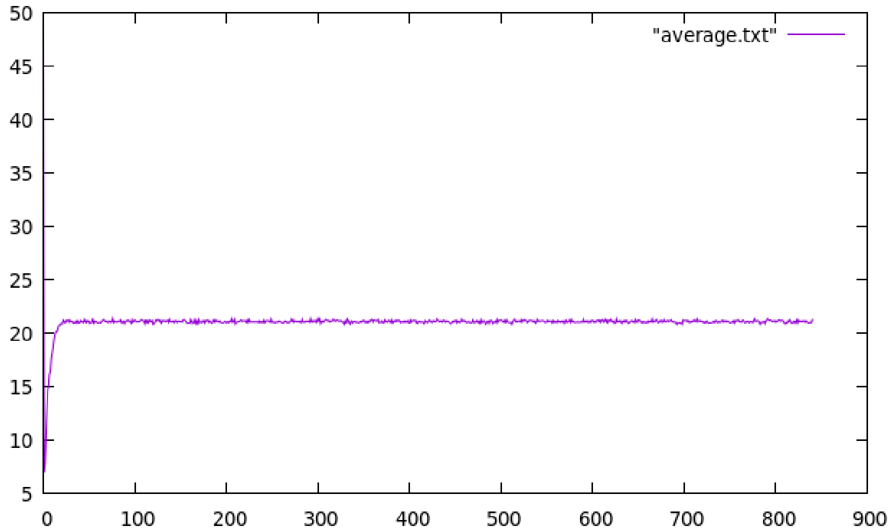

Figura 27: Average live cells

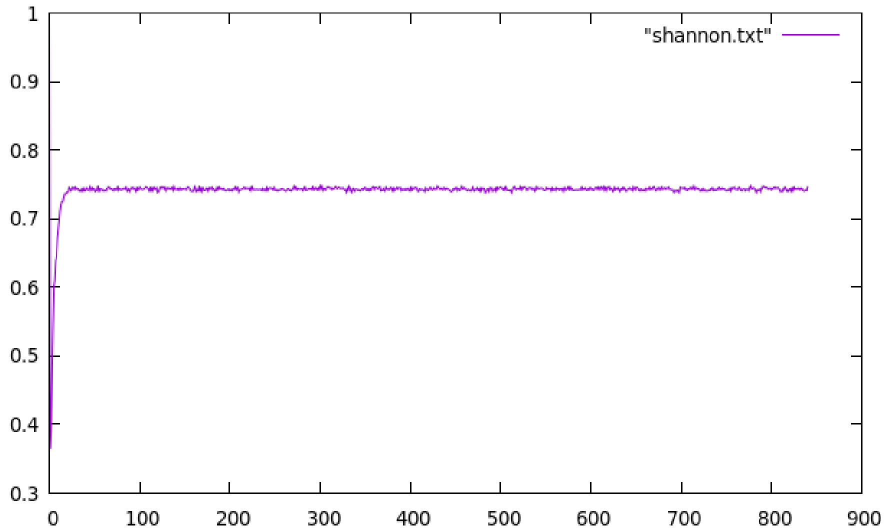

Figura 28: Shannon's Entropy

3.5.2. 2nd Run

System never stabilized. Variance was 9,056,333.8

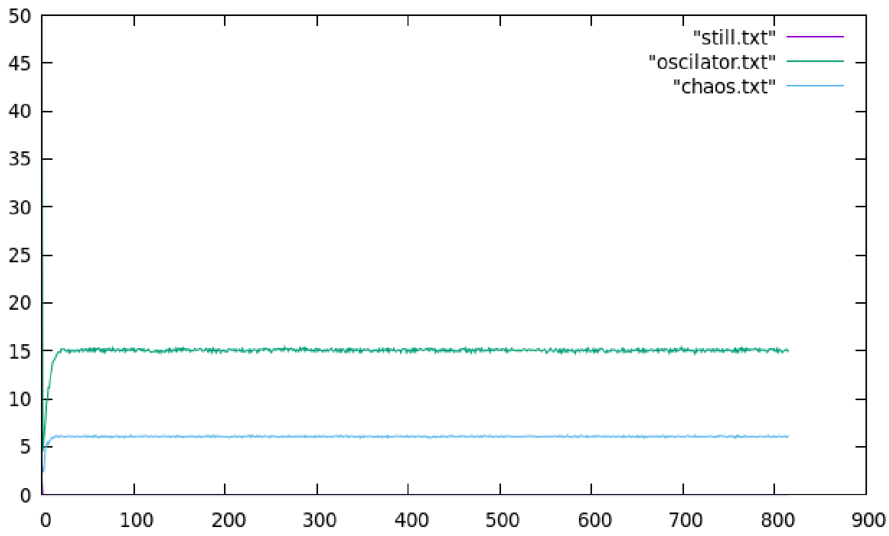

Figura 29: Population Patterns

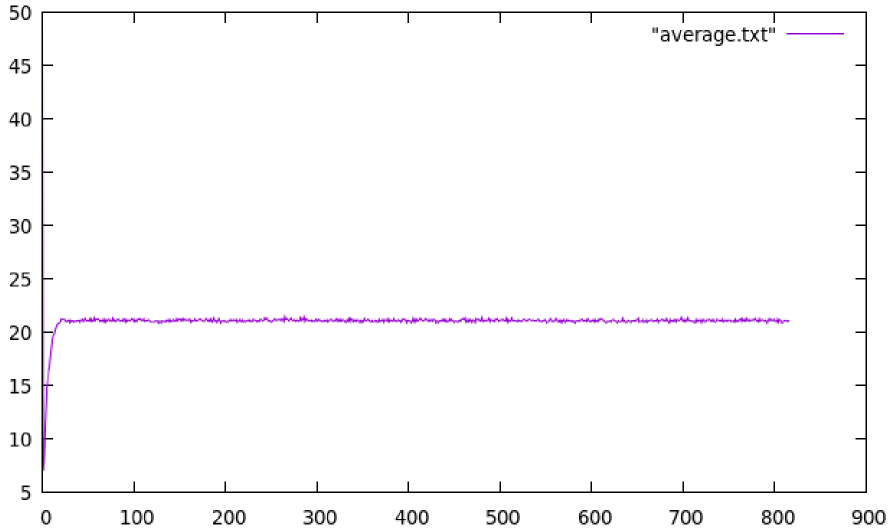

Figura 30: Average live cells

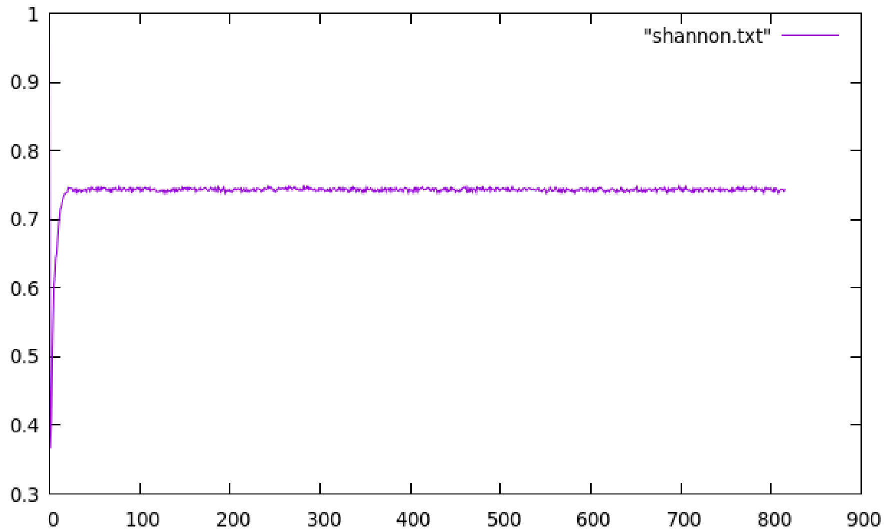

Figura 31: Shannon's Entropy

3.5.3. 3rd Run

System never stabilized. Variance was 11,148,144

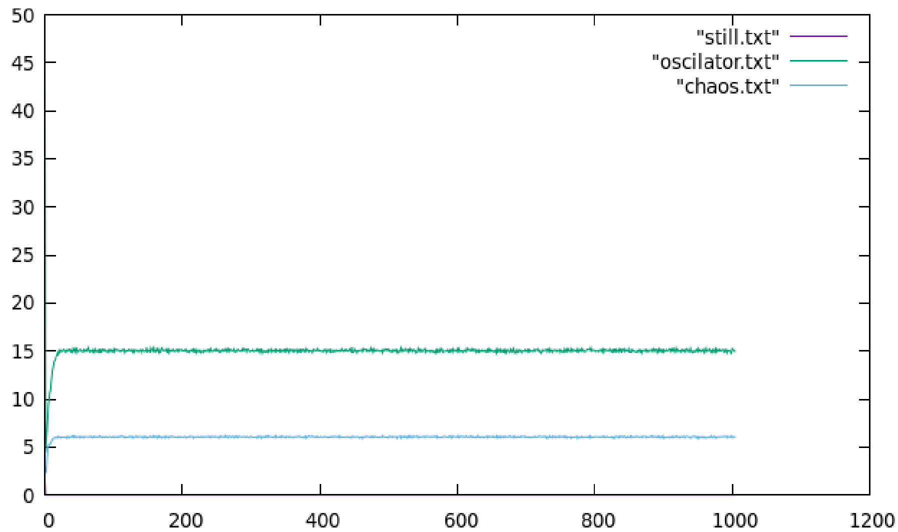

Figura 32: Population Patterns

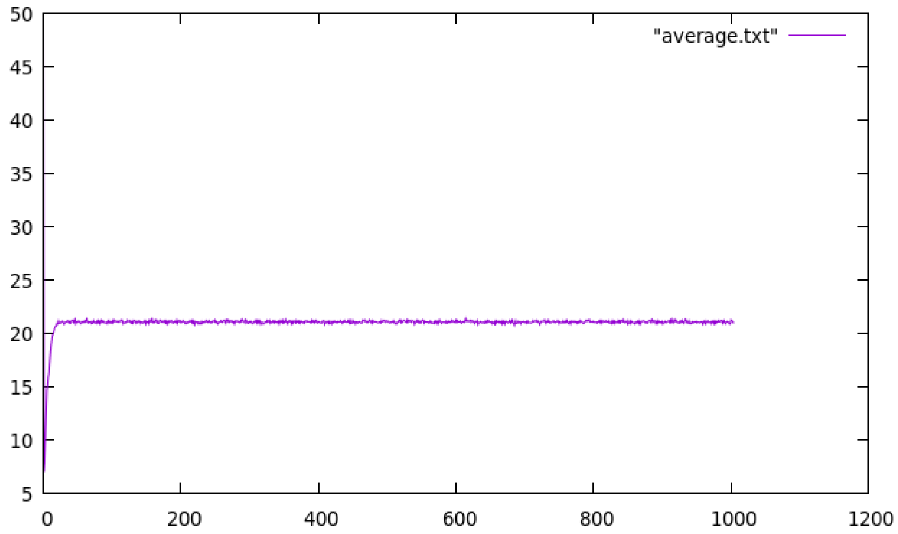

Figura 33: Average live cells

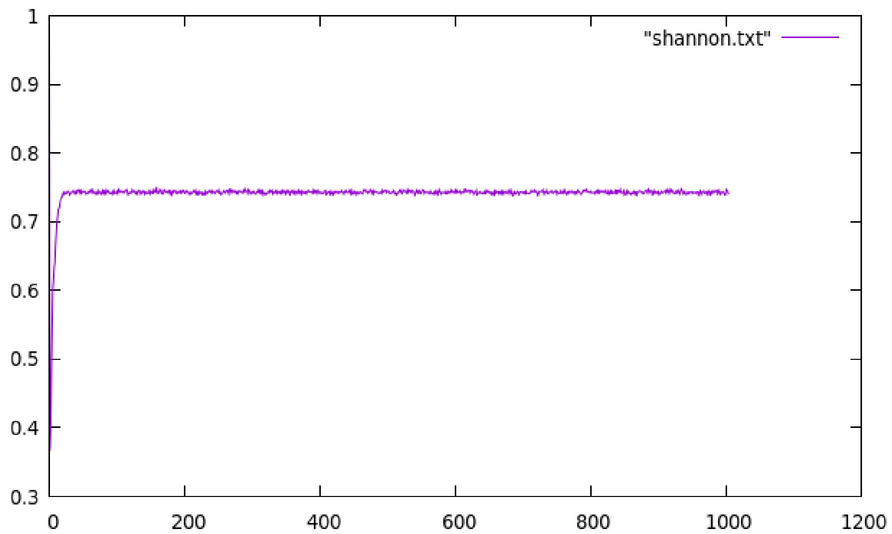

Figura 34: Shannon's Entropy

3.6. Observations: Rule B2/S7 at 50 %

From the three runs we can observe:

- Patterns: The amount of still lives is null. The amount of oscillators and chaotic cells is predominant occupying the 15 % and 5 % and stabilizes immediately after running the simulation.
- The average of live cells immediately drops from 50 % to a value near 20 % and stabilizes immediately after running the simulation.
- The Shannon's Entropy increases quickly after running the simulation and stabilizes at 0.7.

3.7. Rule B2/S7 at 37 %

Testing rule B2/S7 with pattern recognition [Osc-Still-Chaos] activated, up to 10,000 iterations, 37 % live cells at the beginning.

3.7.1. 1st Run

System never stabilized. Variance was 10,523,276

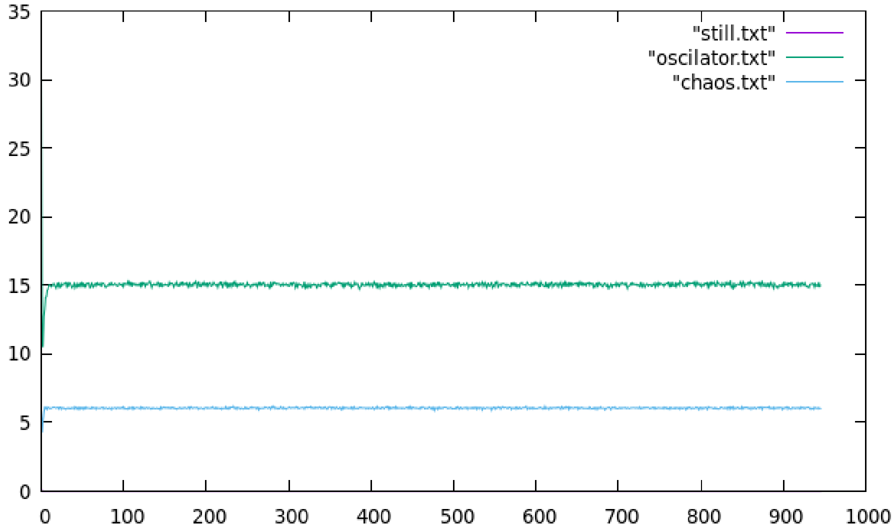

Figura 35: Population Patterns

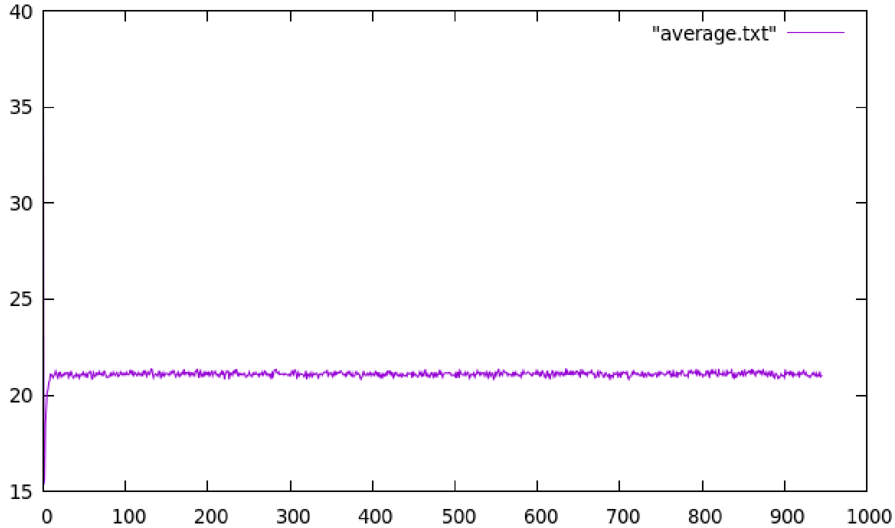

Figura 36: Average live cells

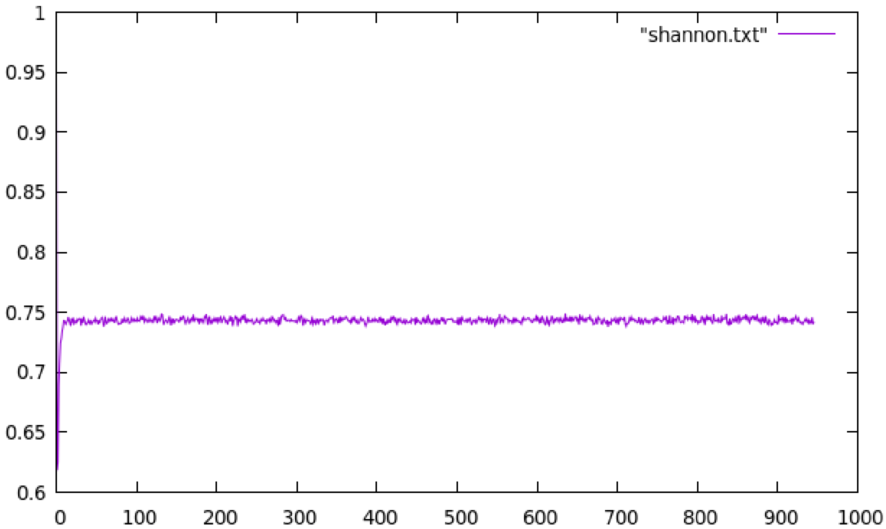

Figura 37: Shannon's Entropy

3.7.2. 2nd Run

System never stabilized. Variance was 10,809,528

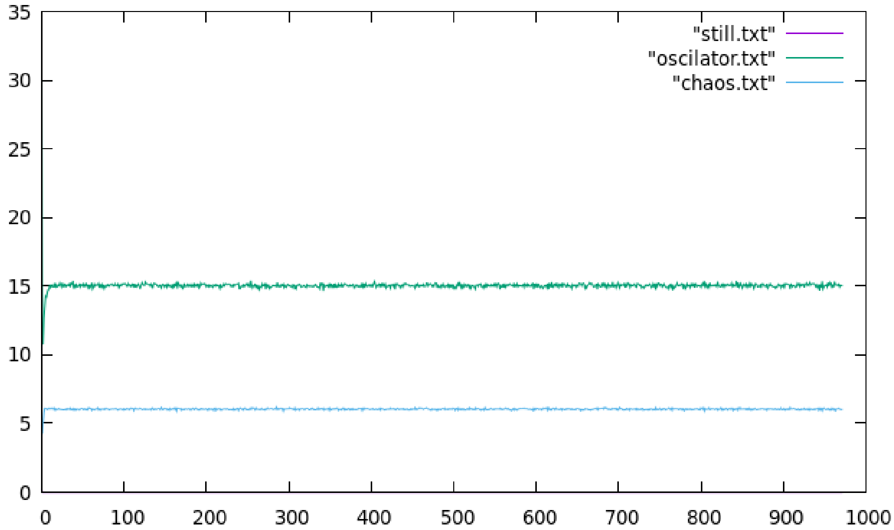

Figura 38: Population Patterns

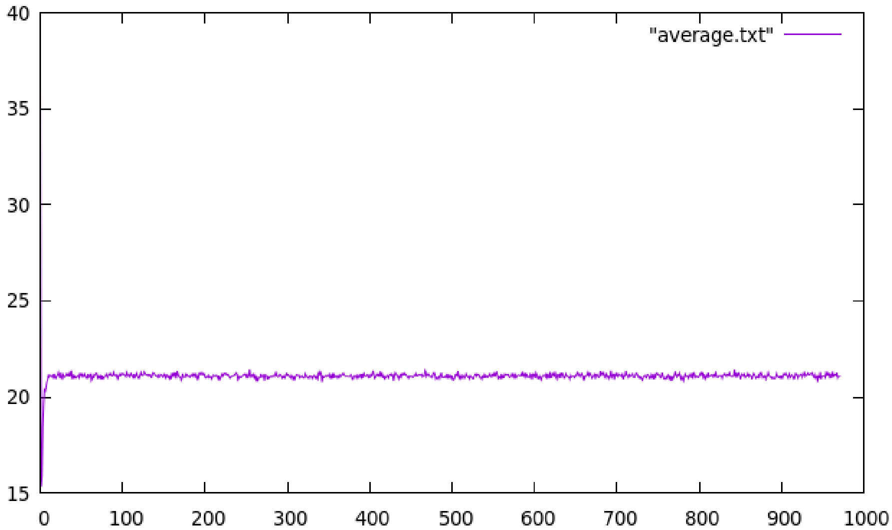

Figura 39: Average live cells

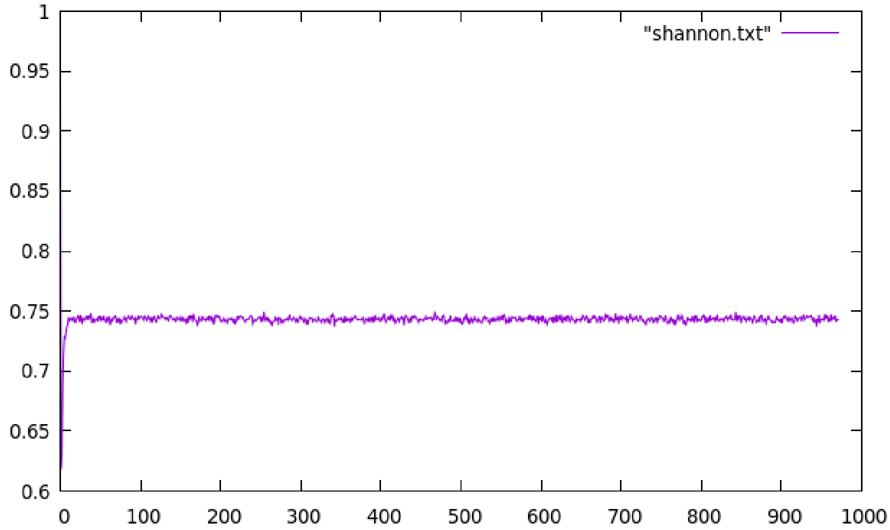

Figura 40: Shannon's Entropy

3.7.3. 3rd Run

System never stabilized. Variance was 11,176,408

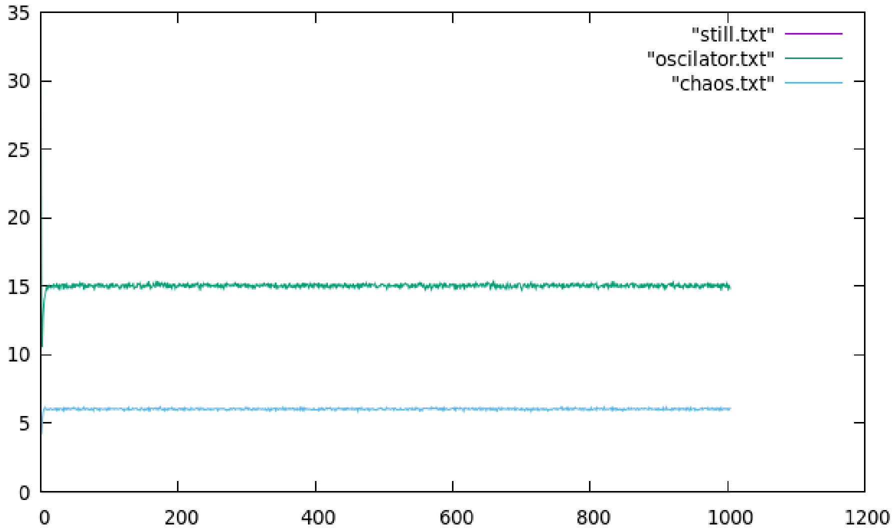

Figura 41: Population Patterns

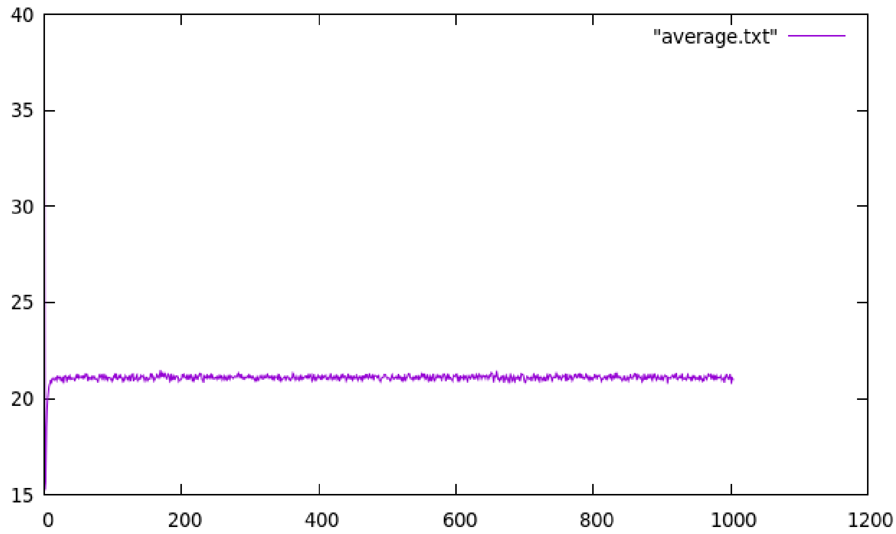

Figura 42: Average live cells

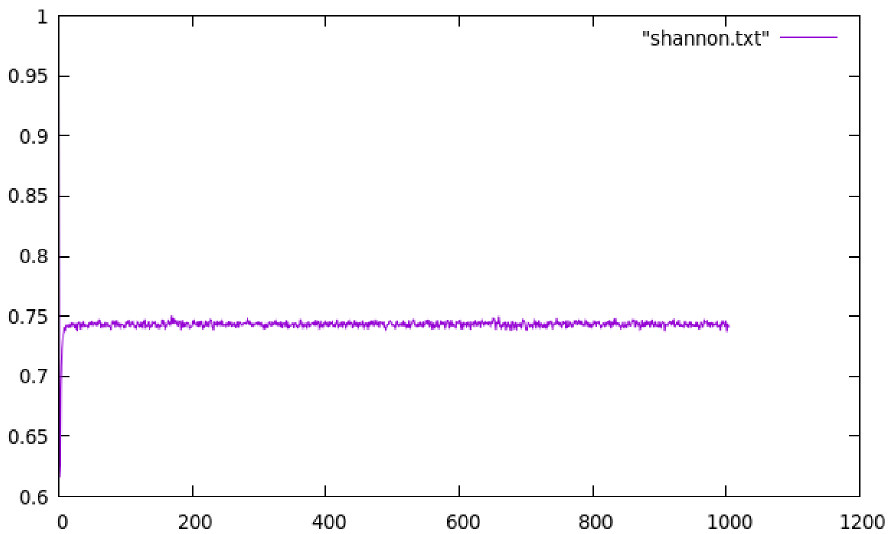

Figura 43: Shannon's Entropy

3.8. Observations: Rule B2/S7 at 37 %

From the three runs we can observe:

- Patterns: The values are the same than with 50 %, there is no difference observable.
- The values are the same than with 50 %, there is no difference observable.
- The Shannon's Entropy behaves equally as with 50 % live cells, but increasing a little bit to 0.75.

3.9. Rule B45678/S3 at 50 %

Testing rule B45678/S3 with no pattern recognition activated, up to 10,000 iterations, 50 % live cells at the beginning.

3.9.1. 1st Run

System stabilized before reaching 100 iterations. Variance was 171,182.6

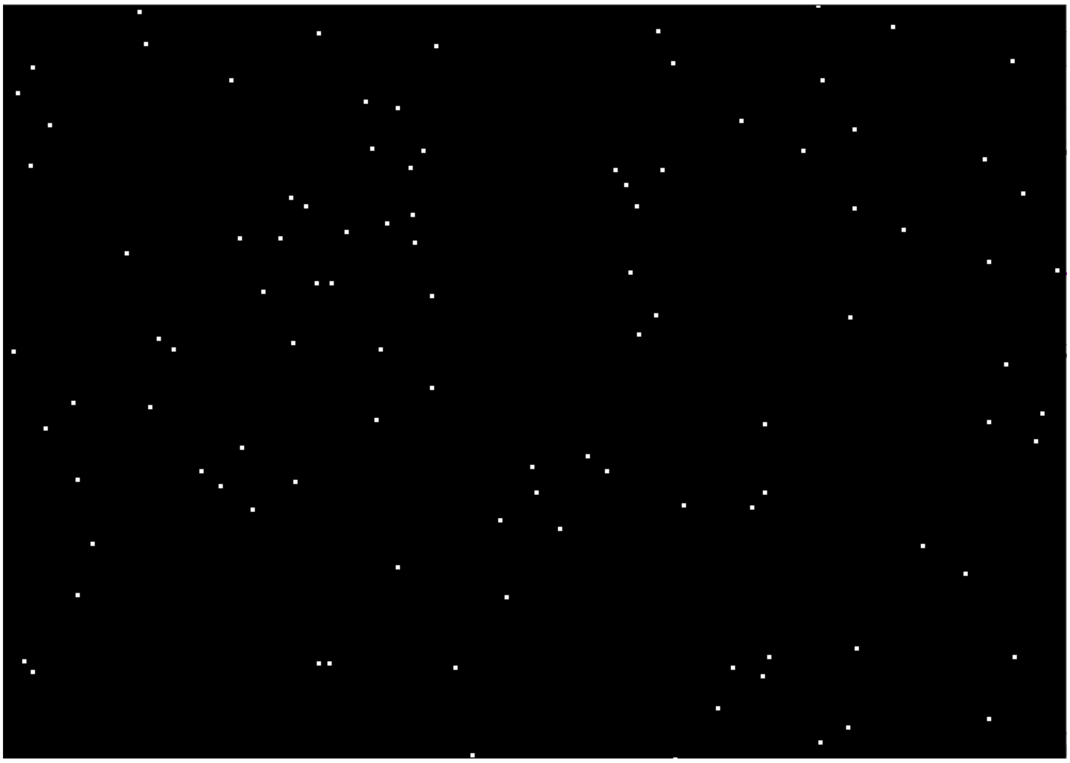

Figura 44: Simulation running

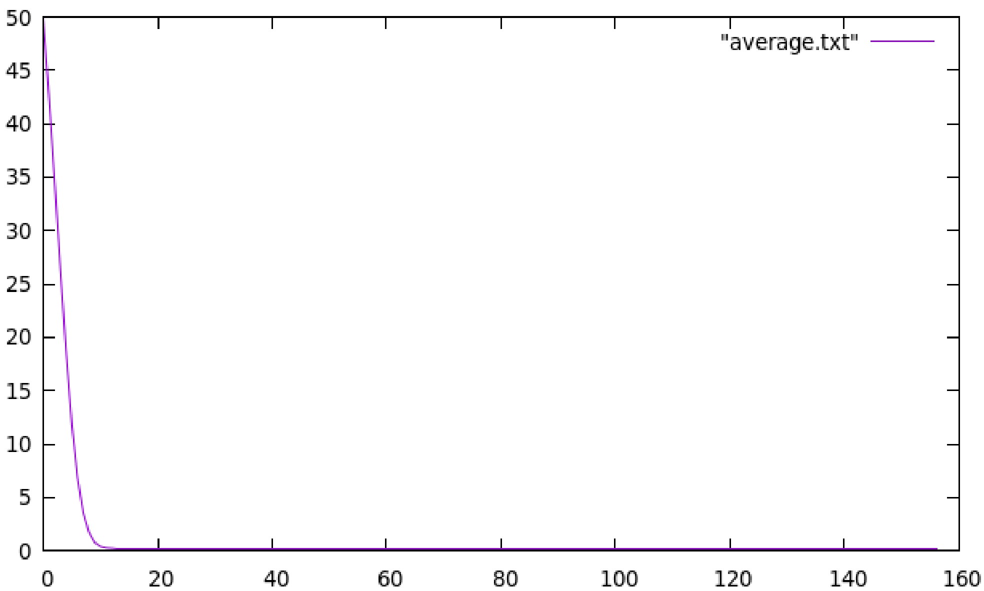

Figura 45: Average live cells

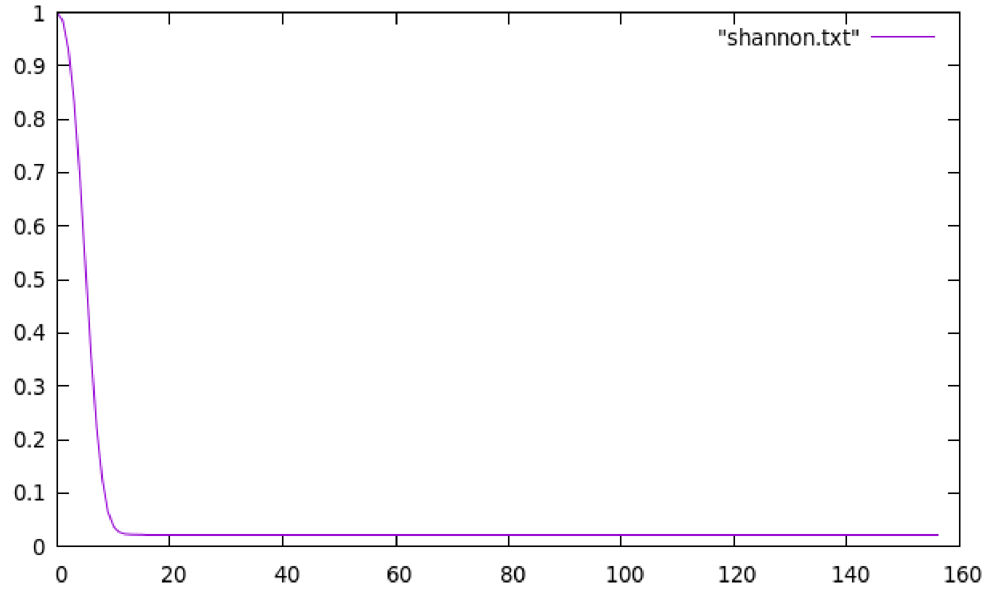

Figura 46: Shannon's Entropy

3.9.2. 2nd Run

System stabilized before reaching 100 iterations. Variance was 171,031.66

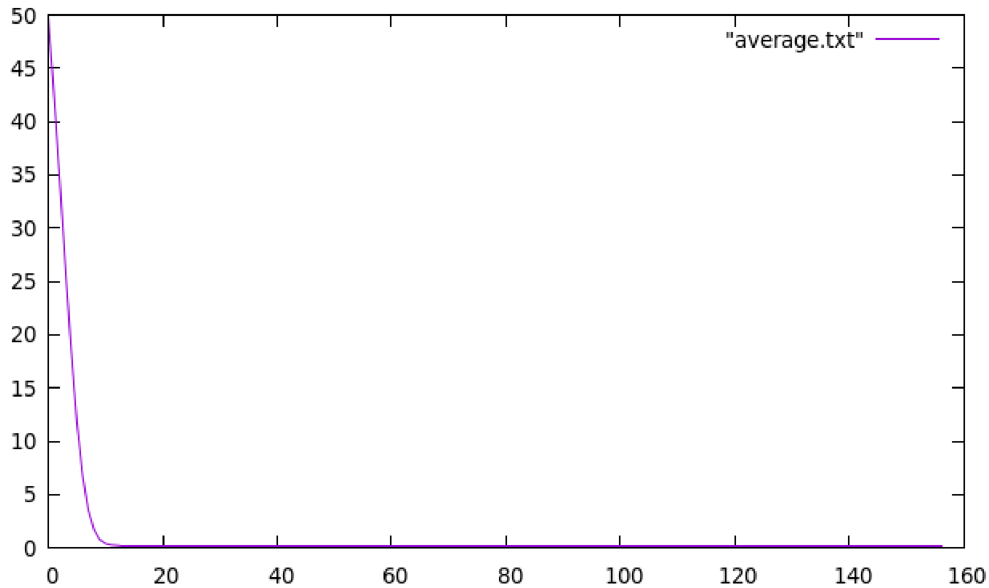

Figura 47: Average live cells

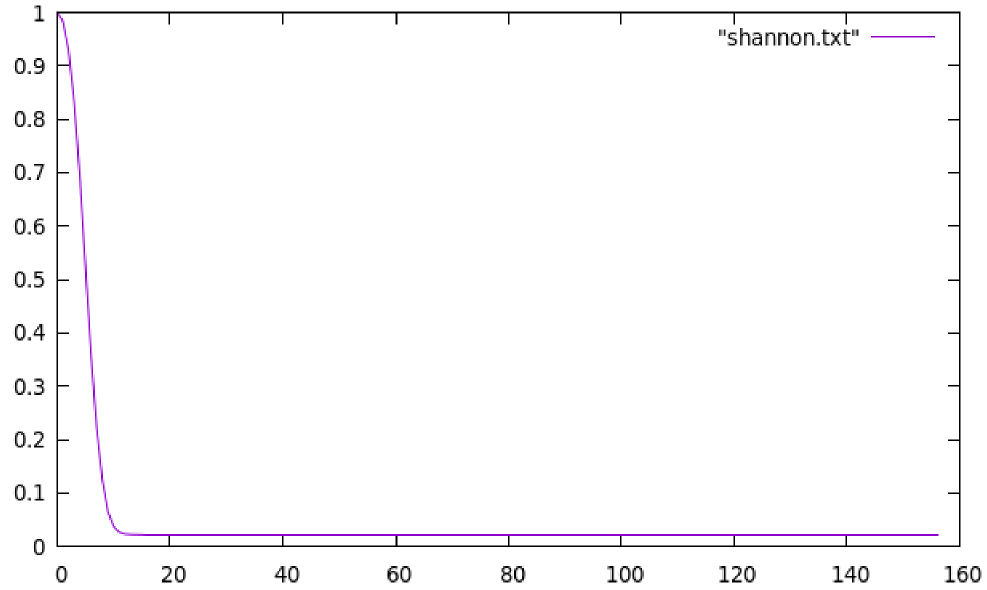

Figura 48: Shannon's Entropy

3.9.3. 3rd Run

System stabilized before reaching 100 iterations. Variance was 171,638.2

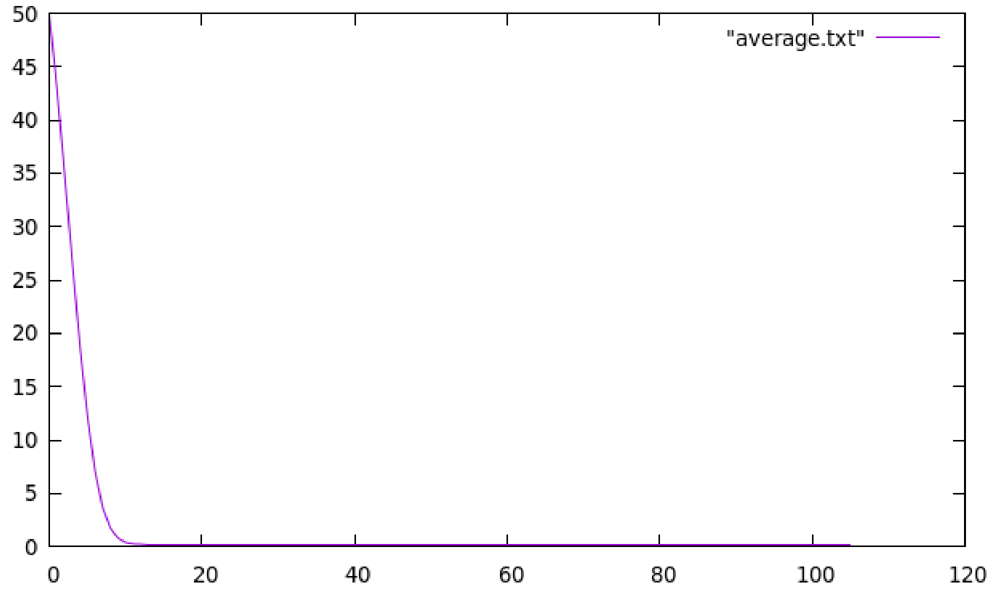

Figura 49: Average live cells

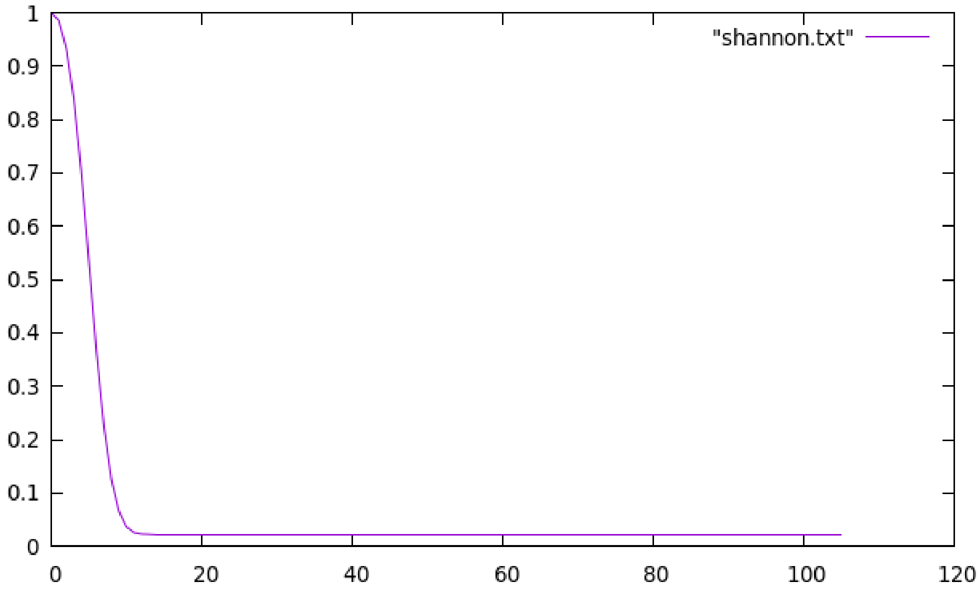

Figura 50: Shannon's Entropy

3.10. Observations: Rule B45678/S3 at 50 %

From the three runs we can observe:

- The amount of live cells drops quickly and even before reaching 20 iterations it stabilizes at almost 0 %.
- The Shannon's Entropy behaves equally reaching almost 0 before 20 iterations.

3.11. Rule B45678/S3 at 37 %

Testing rule B45678/S3 with no pattern recognition activated, up to 10,000 iterations, 37 % live cells at the beginning.

3.11.1. 1st Run

System stabilized before reaching 100 iterations. Variance was 81,817.748

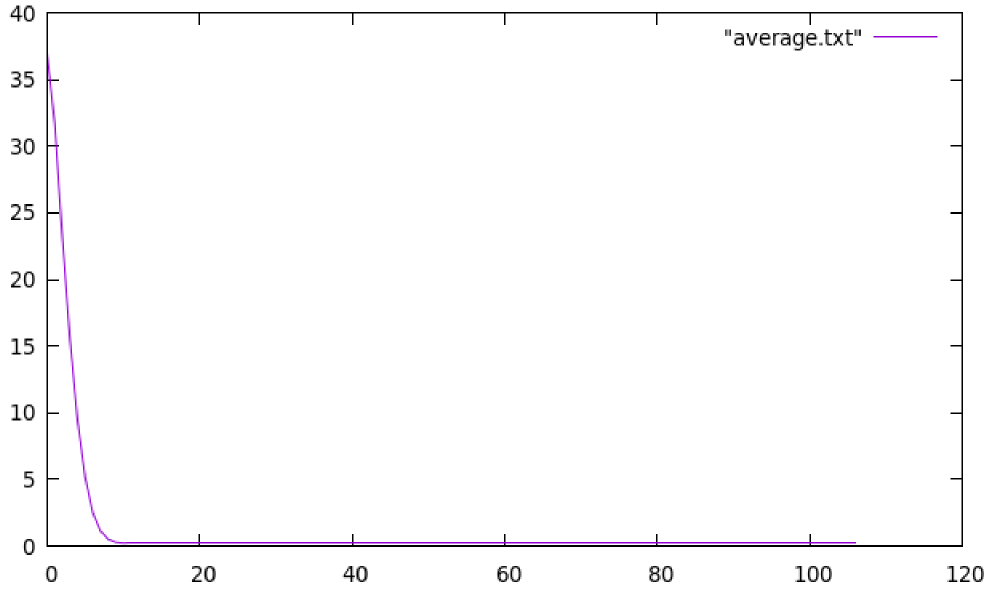

Figura 51: Average live cells

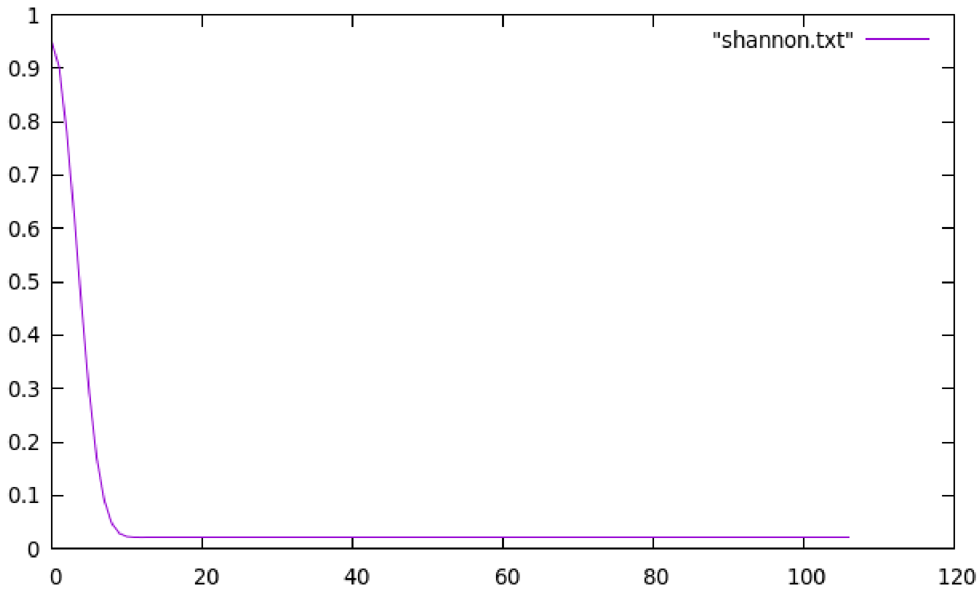

Figura 52: Shannon's Entropy

3.11.2. 2nd Run

System stabilized before reaching 100 iterations. Variance was 82,445.924

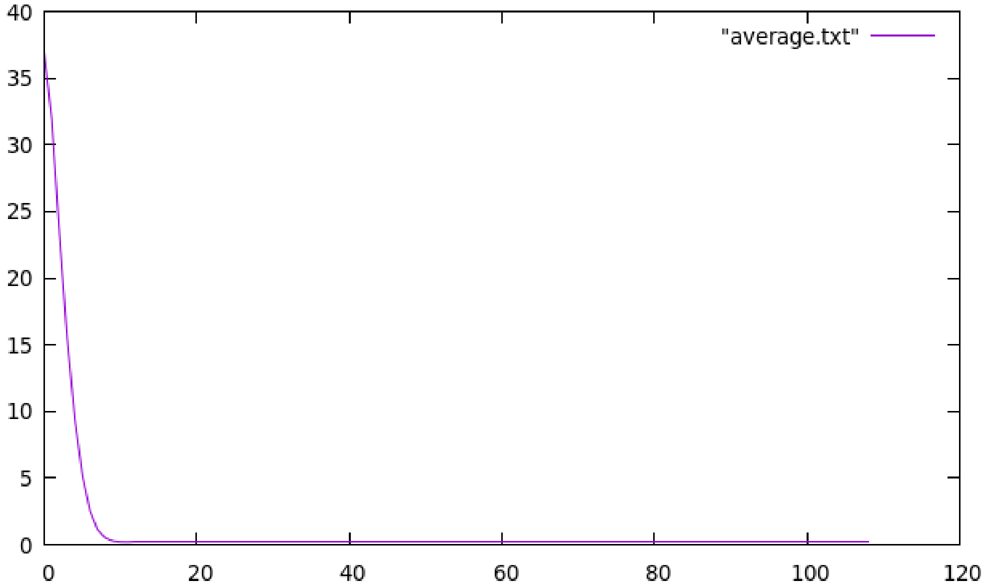

Figura 53: Average live cells

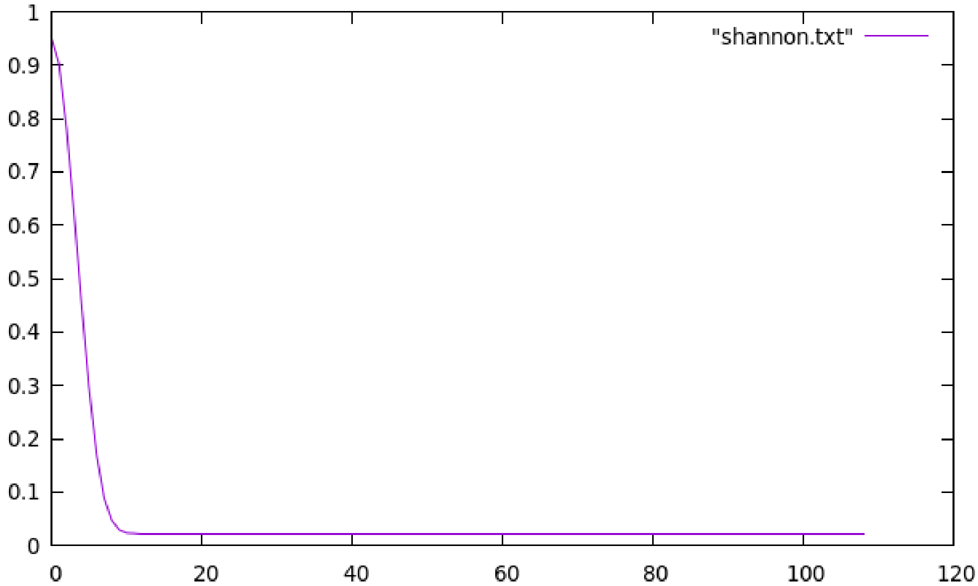

Figura 54: Shannon's Entropy

3.11.3. 3rd Run

System stabilized before reaching 100 iterations. Variance was 82,526.258

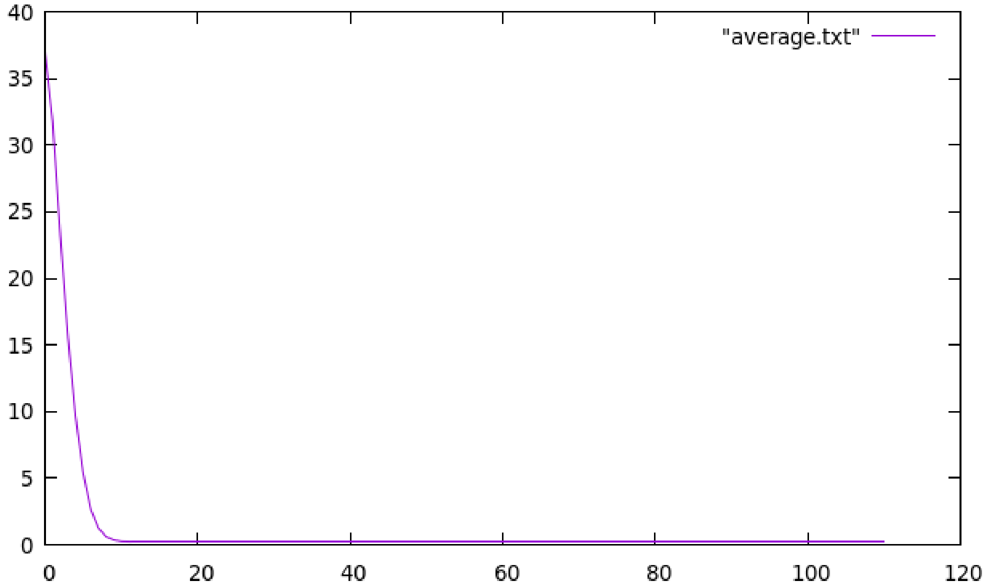

Figura 55: Average live cells

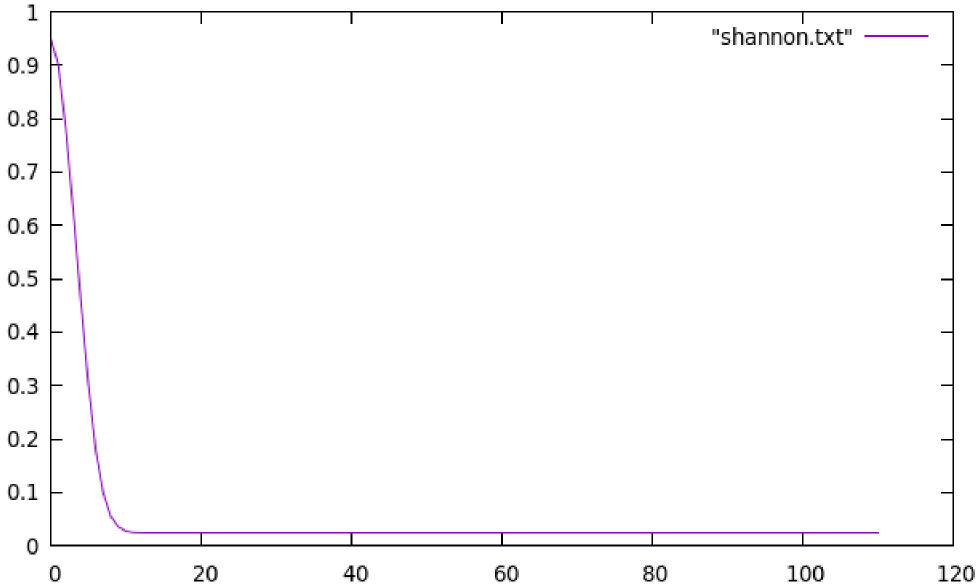

Figura 56: Shannon's Entropy

3.12. Observations: Rule B45678/S3 at 37 %

From the three runs we can observe that it behaves almost exactly the same as with 50 % live cells.

3.13. Rule B123456/S123456 at 50 %

Testing rule B123456/S123456 with no pattern recognition activated, up to 10,000 iterations, 50 % live cells at the beginning.

3.13.1. 1st Run

System never stabilized. Variance was 7,086,196.2

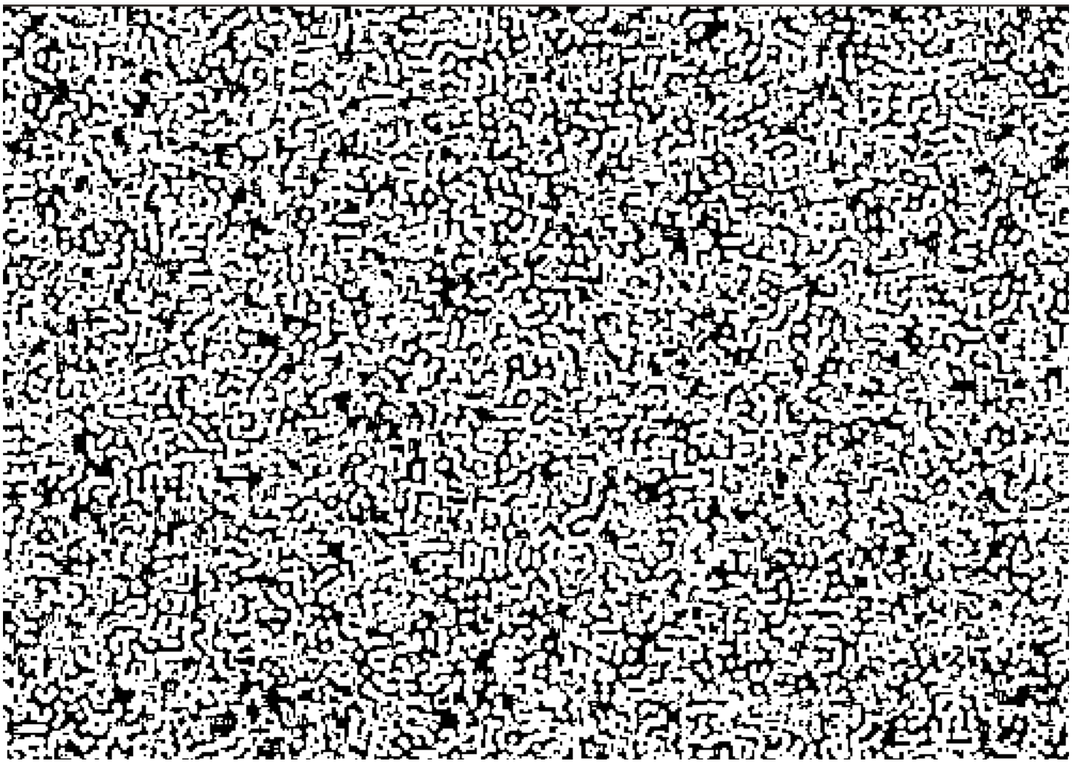

Figura 57: Simulation running

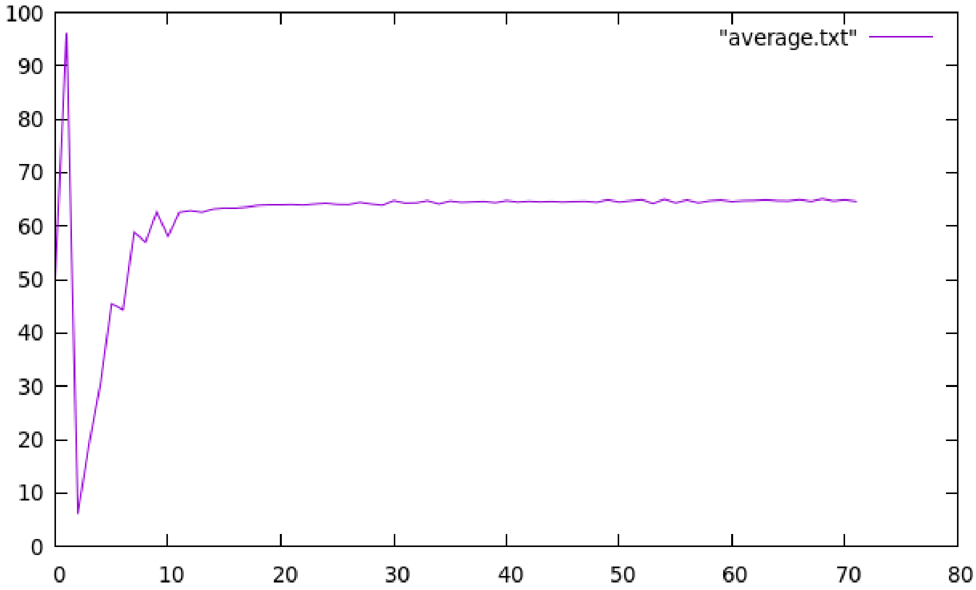

Figura 58: Average live cells

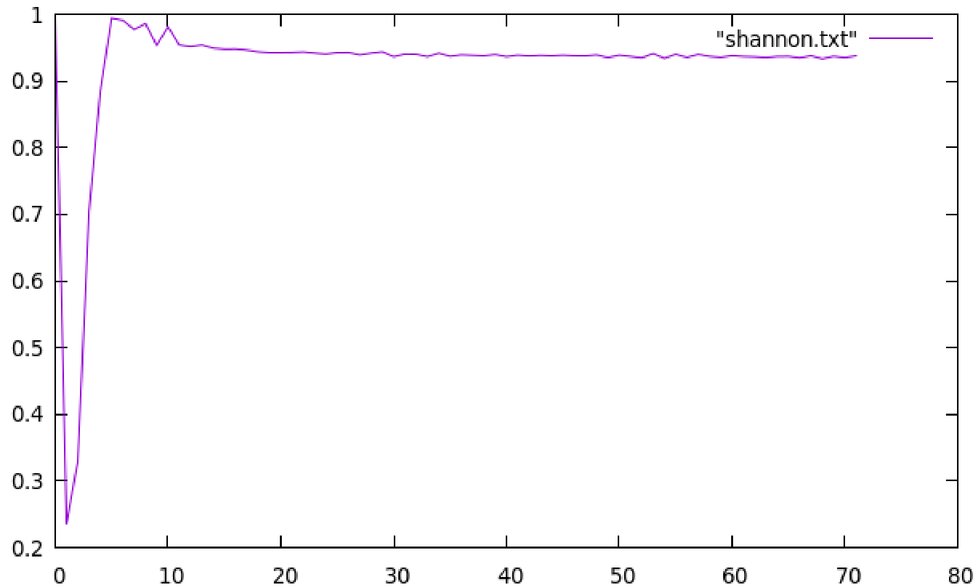

Figura 59: Shannon's Entropy

3.13.2. 2nd Run

System never stabilized. Variance was 5,405,346.9

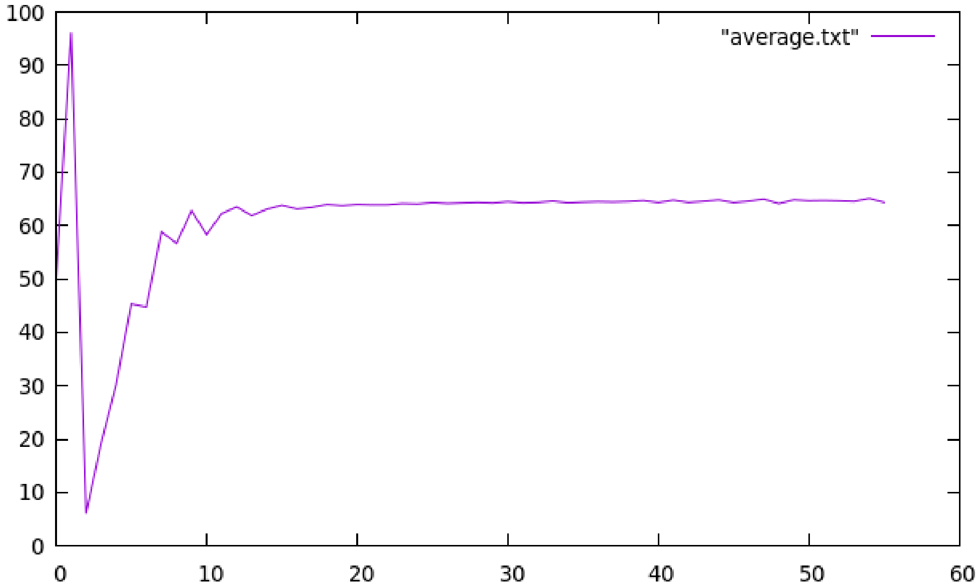

Figura 60: Average live cells

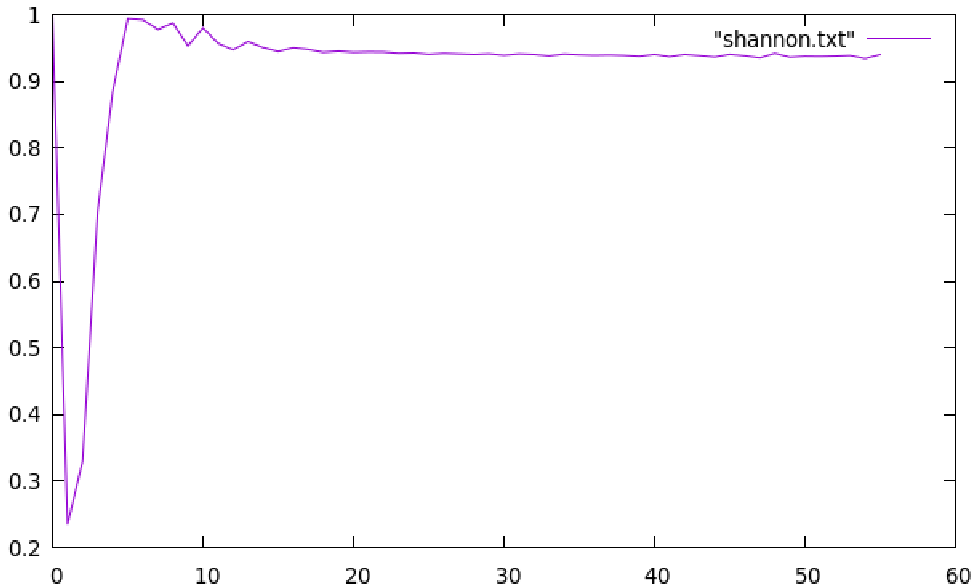

Figura 61: Shannon's Entropy

3.13.3. 3rd Run

System never stabilized. Variance was 5,734,793.7

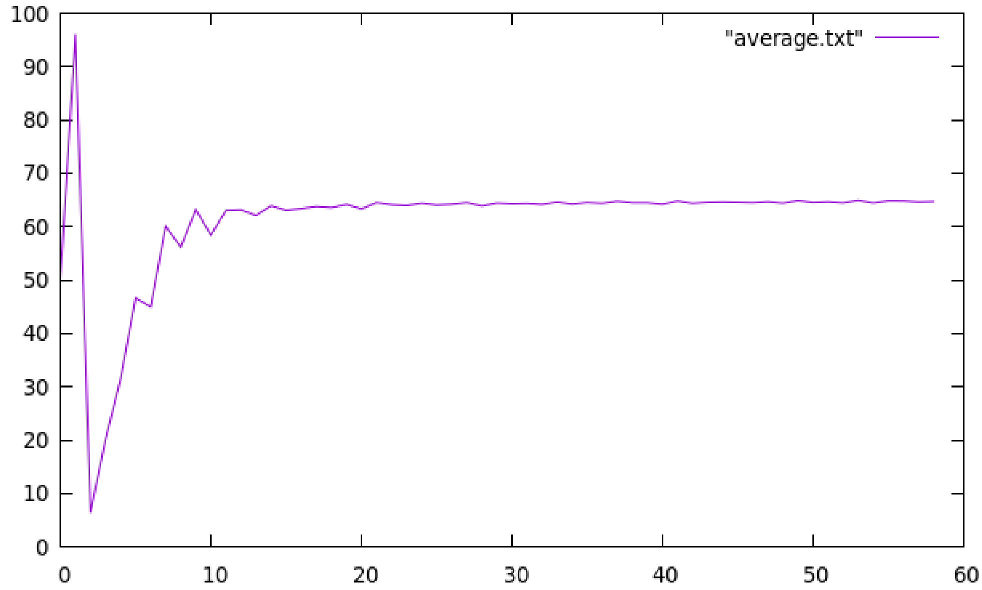

Figura 62: Average live cells

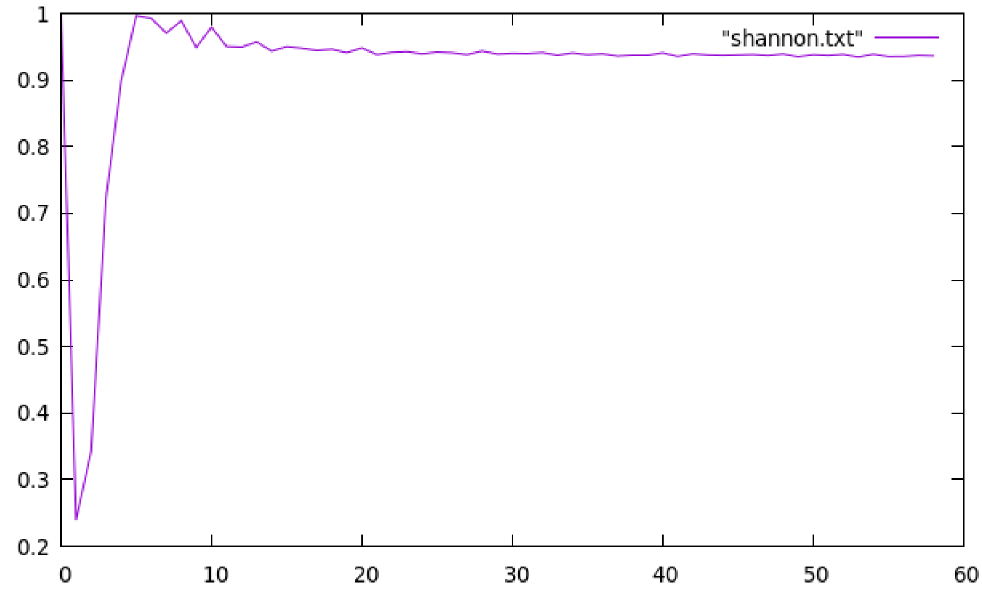

Figura 63: Shannon's Entropy

3.14. Observations: Rule B123456/S123456 at 50 %

From the three runs we can observe:

- The amount of live cells increases fast and drops quickly and even before reaching 20 iterations it stabilizes at almost 60 %.
- The Shannon's Entropy drops fast the it increases reaching 1.0 and then stabilizes above 0.9.

3.15. Rule B123456/S123456 at 37 %

Testing rule B123456/S123456 with no pattern recognition activated, up to 10,000 iterations, 37 % live cells at the beginning.

3.15.1. 1st Run

System never stabilized. Variance was 5,277,586.7

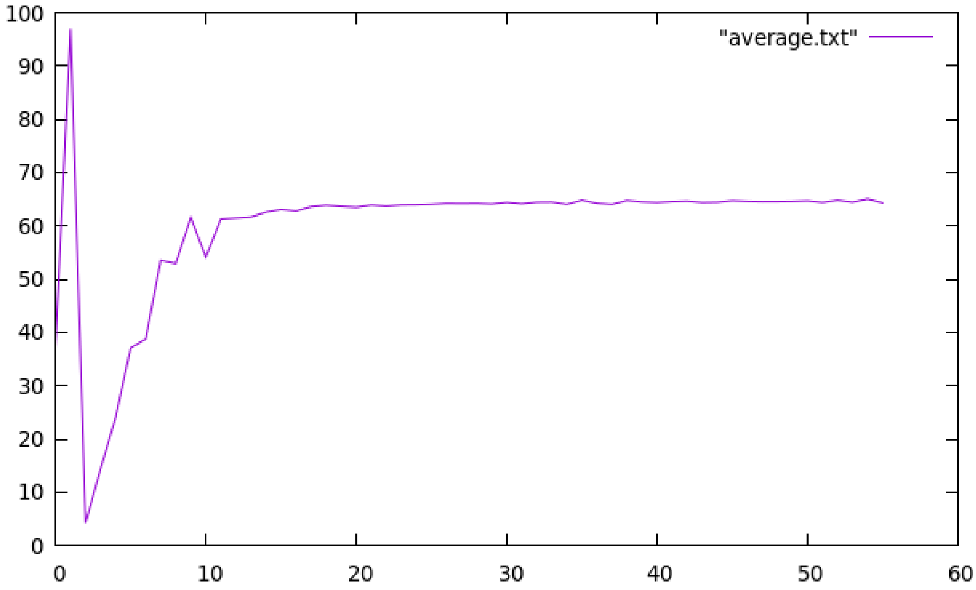

Figura 64: Average live cells

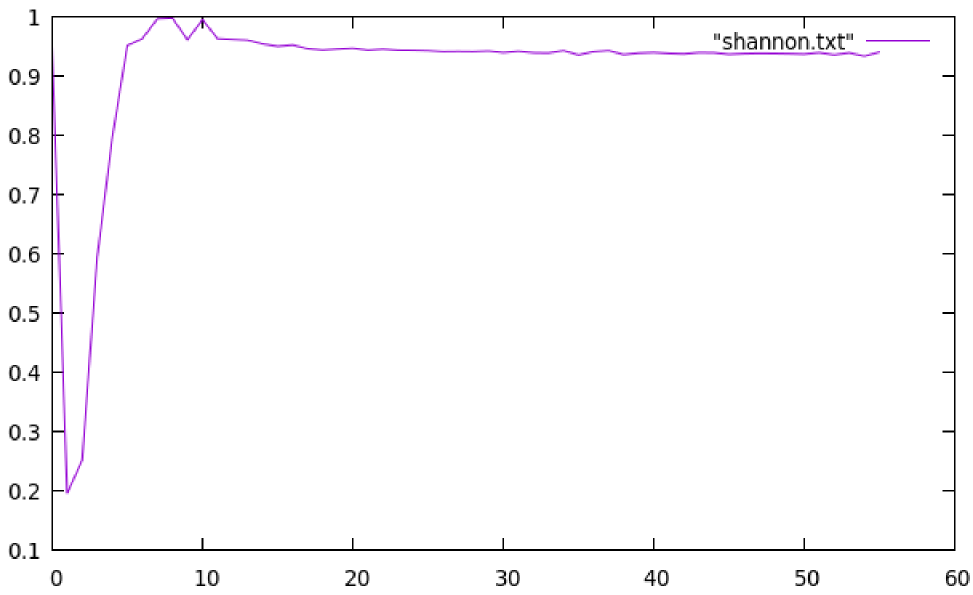

Figura 65: Shannon's Entropy

3.15.2. 2nd Run

System never stabilized. Variance was 5,064,228.6

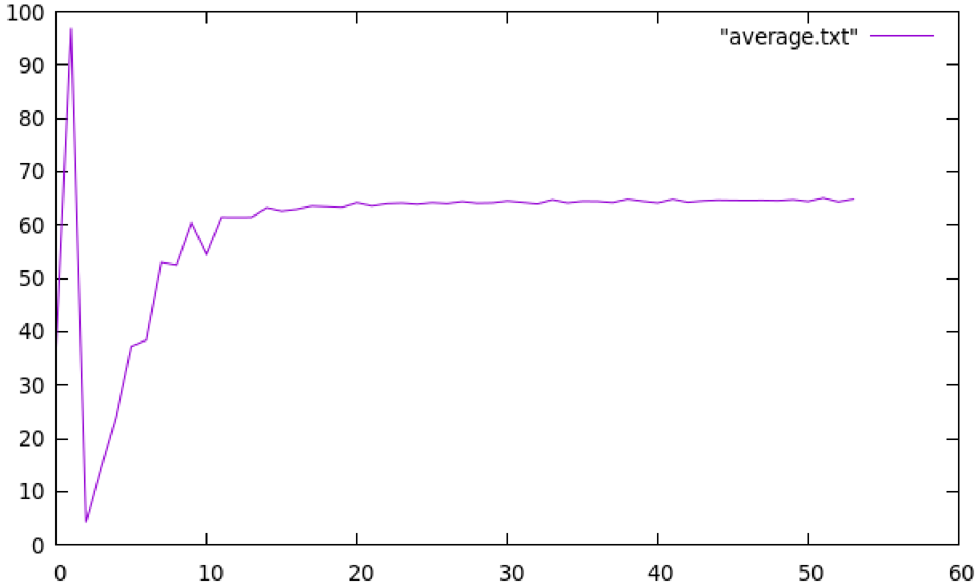

Figura 66: Average live cells

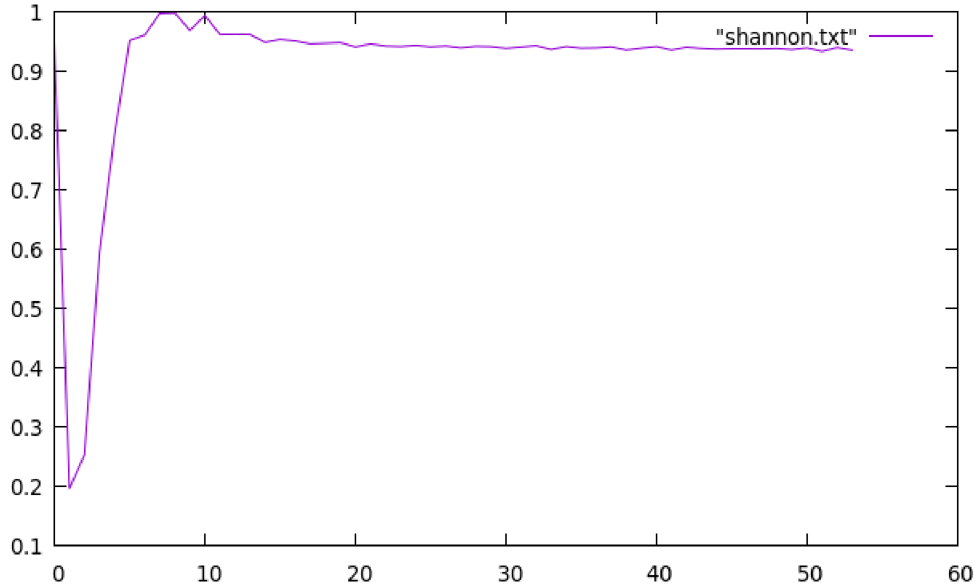

Figura 67: Shannon's Entropy

3.15.3. 3rd Run

System never stabilized. Variance was 5,174,326

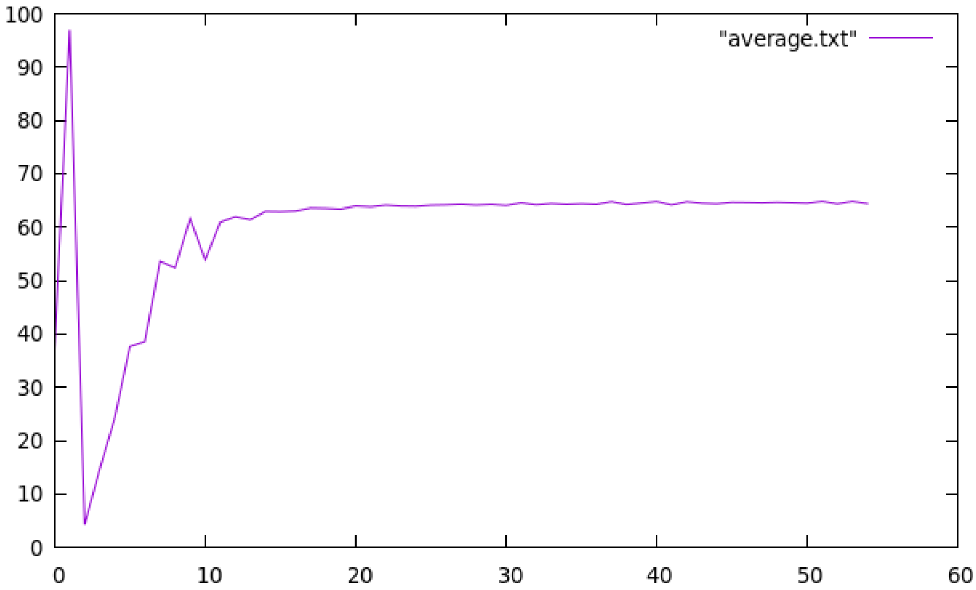

Figura 68: Average live cells

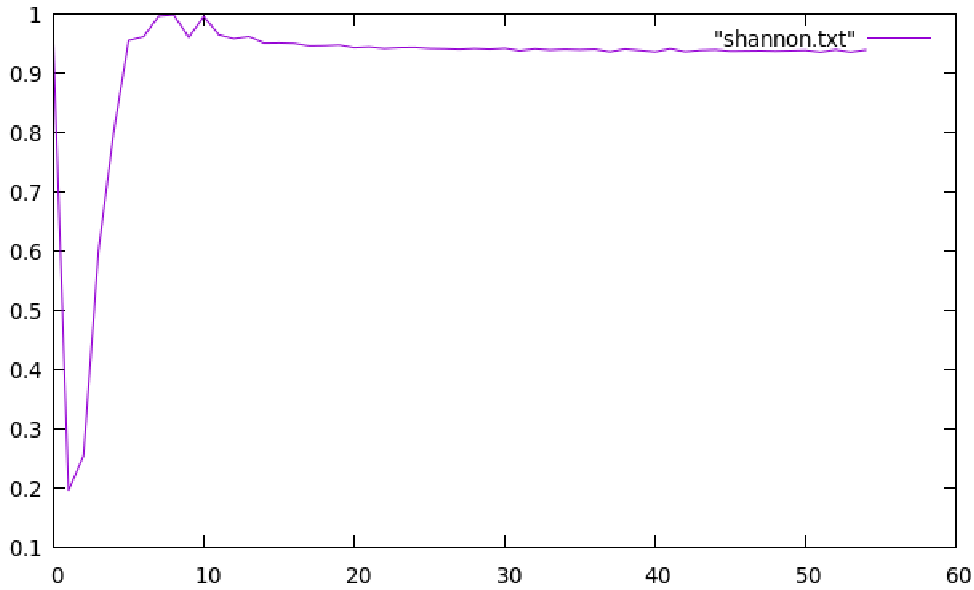

Figura 69: Shannon's Entropy

3.16. Observations: Rule B123456/S123456 at 37 %

From the three runs we can conclude that theres is no significant observable change from the configuration with 50 % live cells.

3.17. Rule B3/S12345678 at 50 %

Testing rule B3/S12345678 with no pattern recognition activated, up to 10,000 iterations, 50 % live cells at the beginning.

3.17.1. 1st Run

System stabilized before reaching 100 iterations. Variance was 10,694,861.

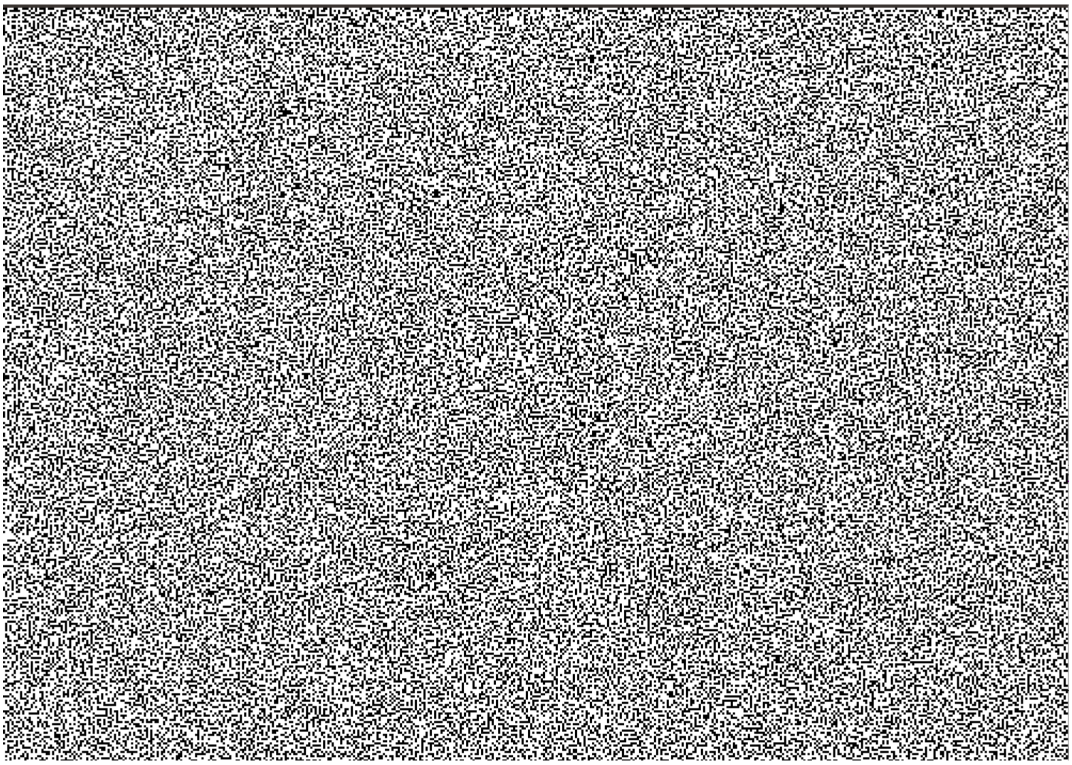

Figura 70: Simulation running

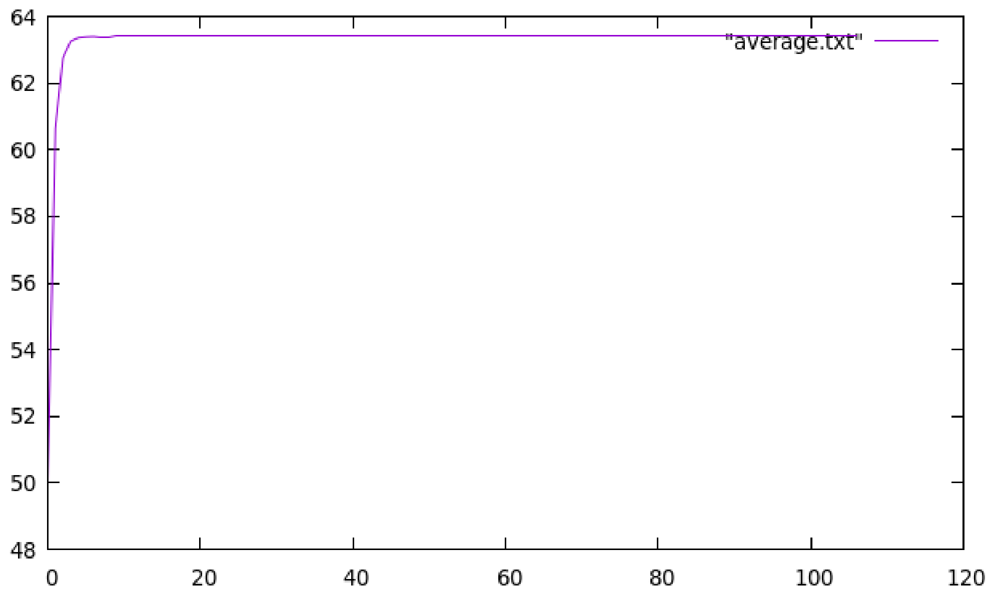

Figura 71: Average live cells

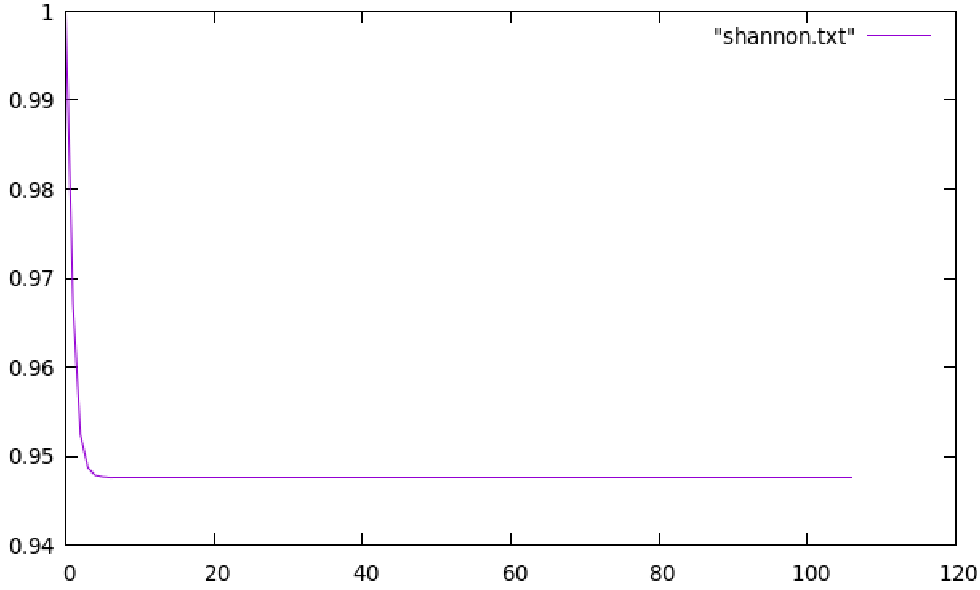

Figura 72: Shannon's Entropy

3.17.2. 2nd Run

System stabilized before reaching 100 iterations. Variance was 1,0745,114.

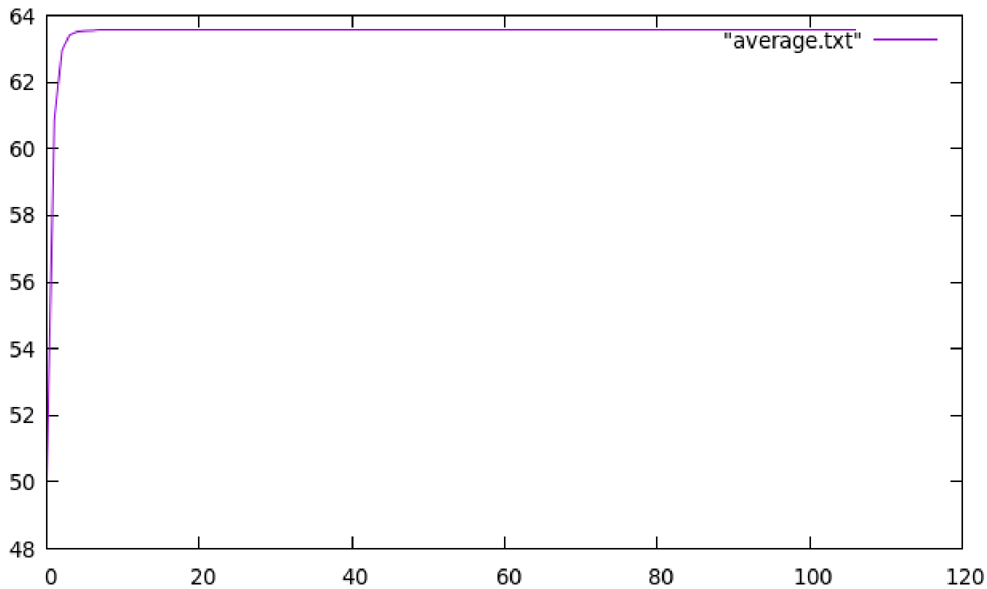

Figura 73: Average live cells

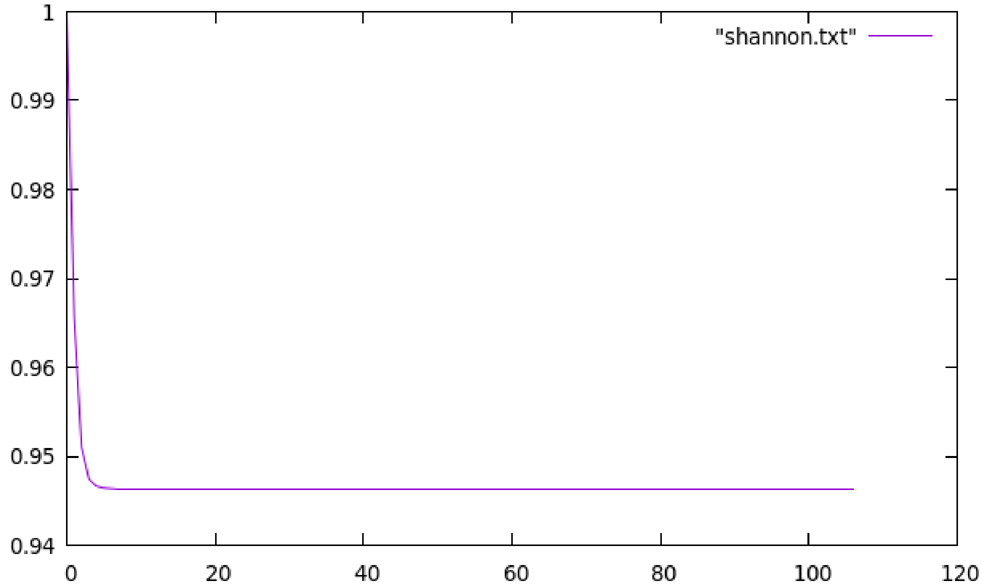

Figura 74: Shannon's Entropy

3.17.3. 3rd Run

System stabilized before reaching 100 iterations. Variance was 11,119,387.

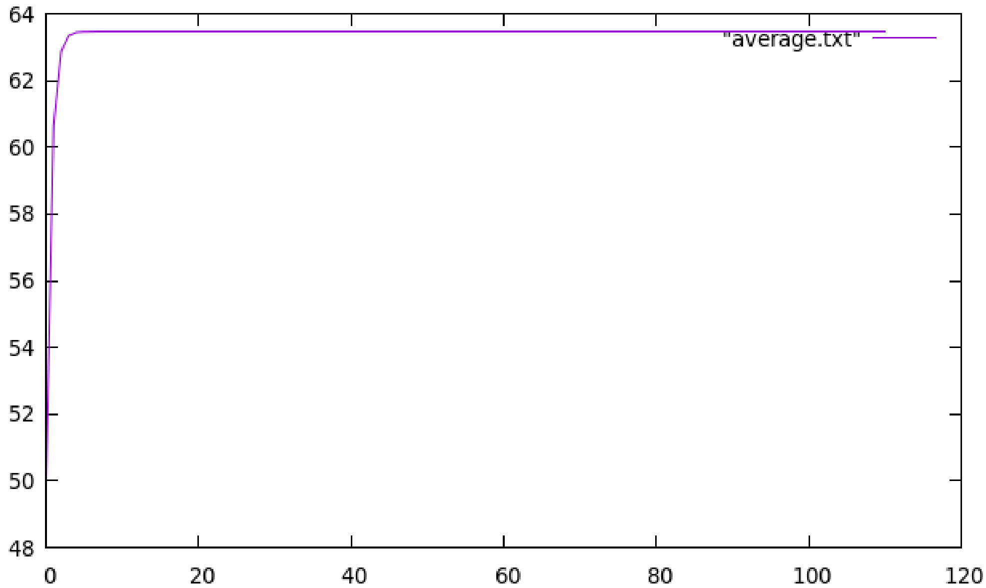

Figura 75: Average live cells

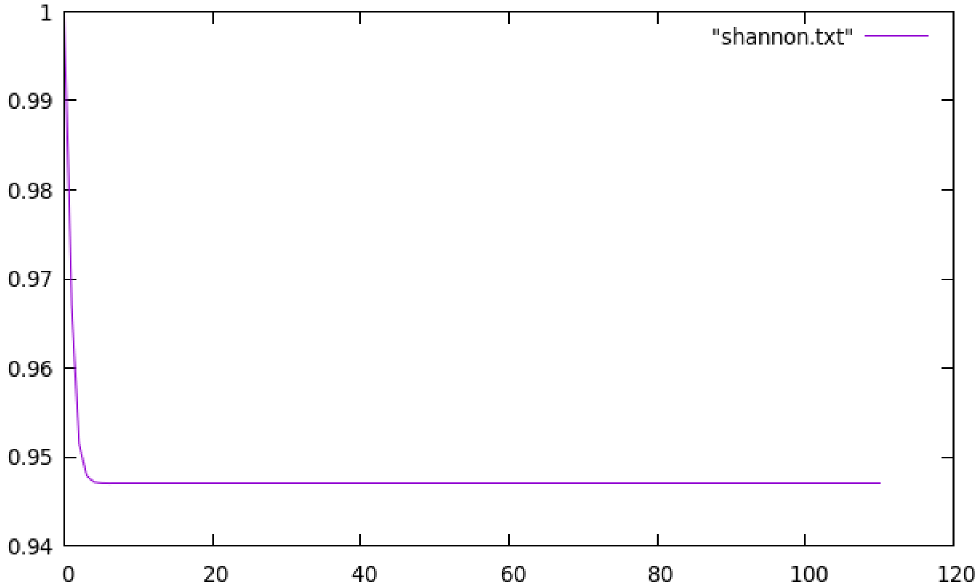

Figura 76: Shannon's Entropy

3.18. Observations: Rule B3/S12345678 at 50 %

From the three runs we can observe:

- The amount of live cells increases fast and even before reaching 20 iterations it stabilizes at almost 63 %.
- The Shannon's Entropy drops fast reaching 0.95 and stabilizes there.

3.19. Rule B3/S12345678 at 37 %

Testing rule B3/S12345678 with no pattern recognition activated, up to 10,000 iterations, 37 % live cells at the beginning.

3.19.1. 1st Run

System stabilized before reaching 100 iterations. Variance was 11,582,671.

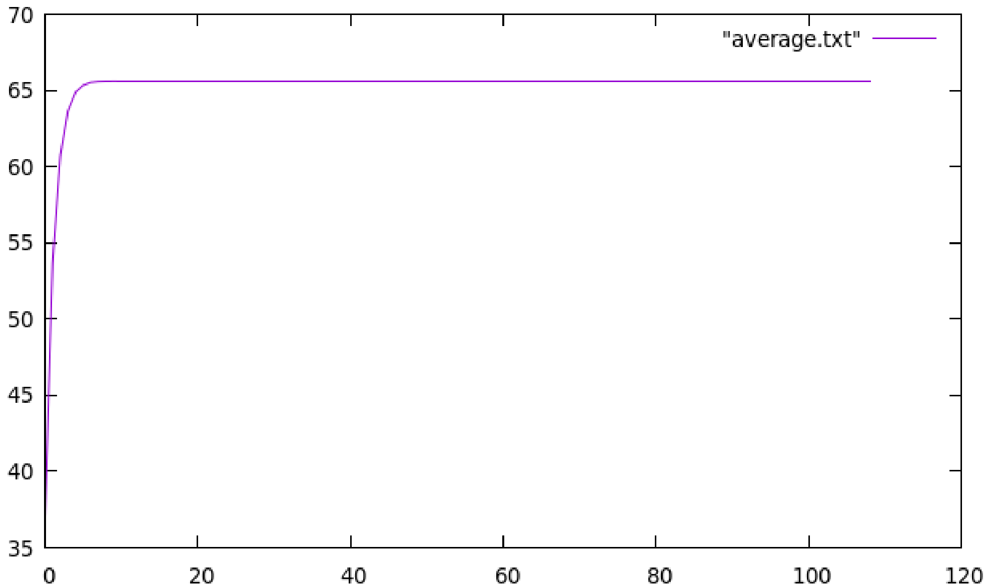

Figura 77: Average live cells

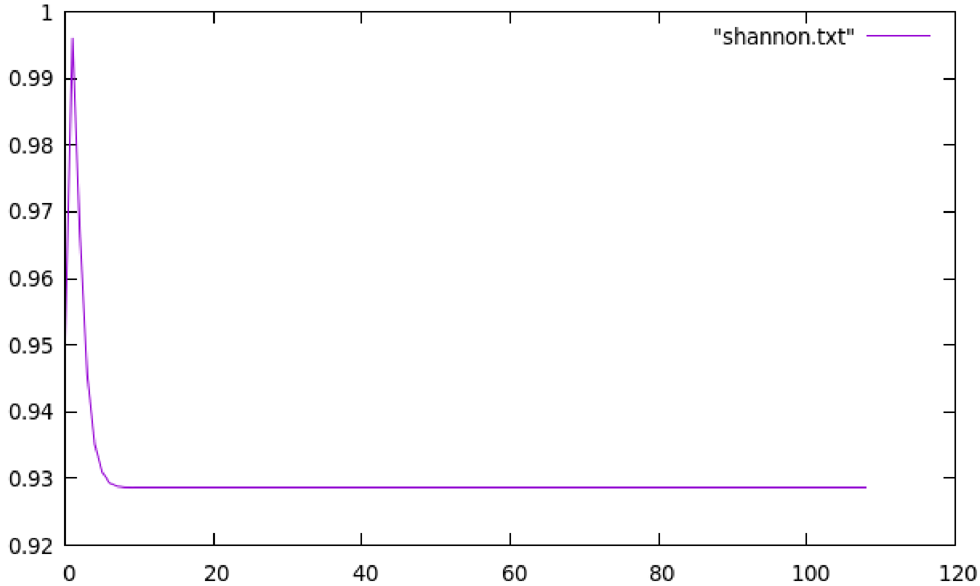

Figura 78: Shannon's Entropy

3.19.2. 2nd Run

System stabilized before reaching 100 iterations. Variance was 11,719,666.

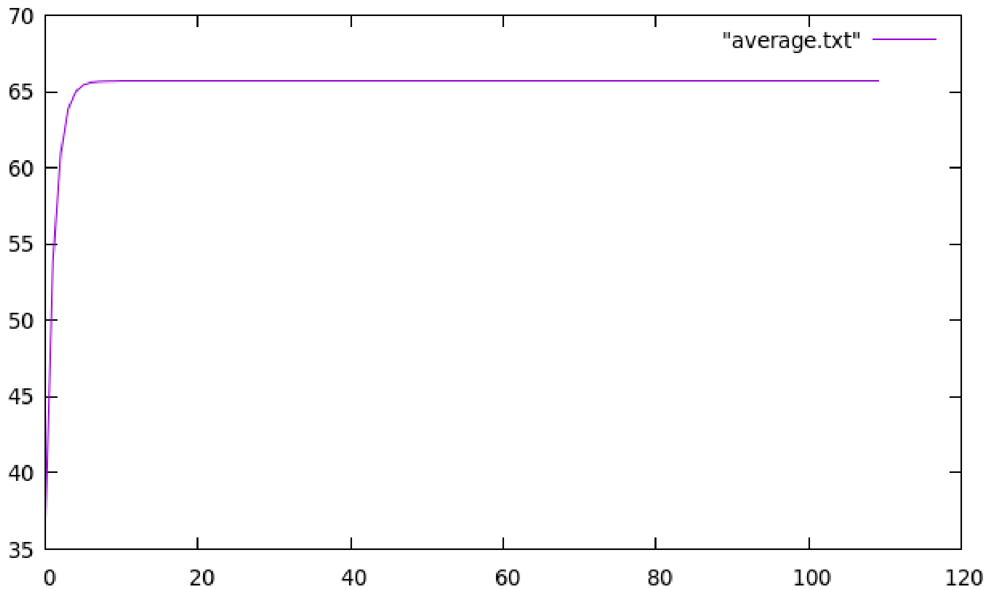

Figura 79: Average live cells

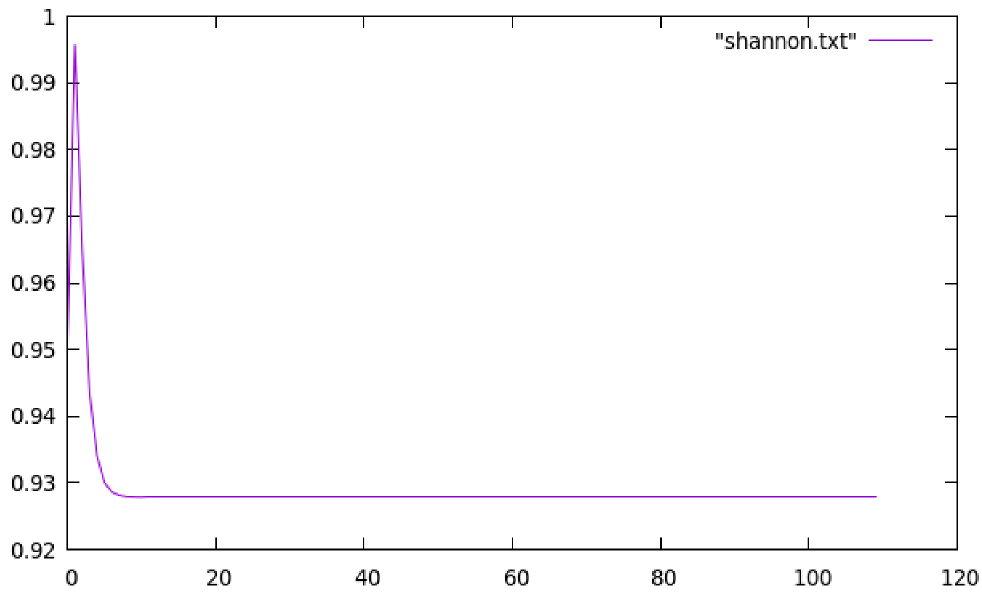

Figura 80: Shannon's Entropy

3.19.3. 3rd Run

System stabilized before reaching 100 iterations. Variance was 12,116,238.

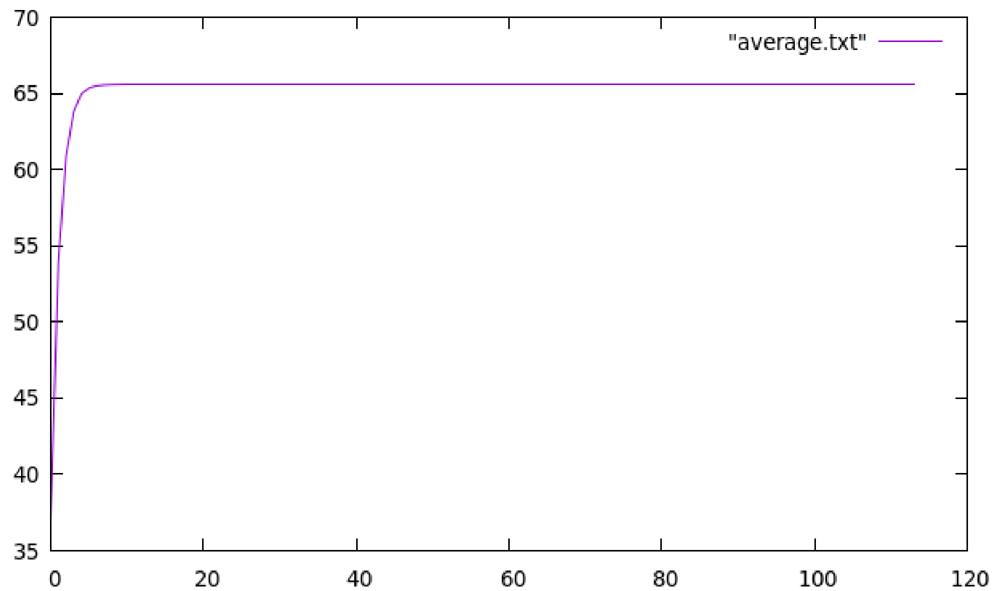

Figura 81: Average live cells

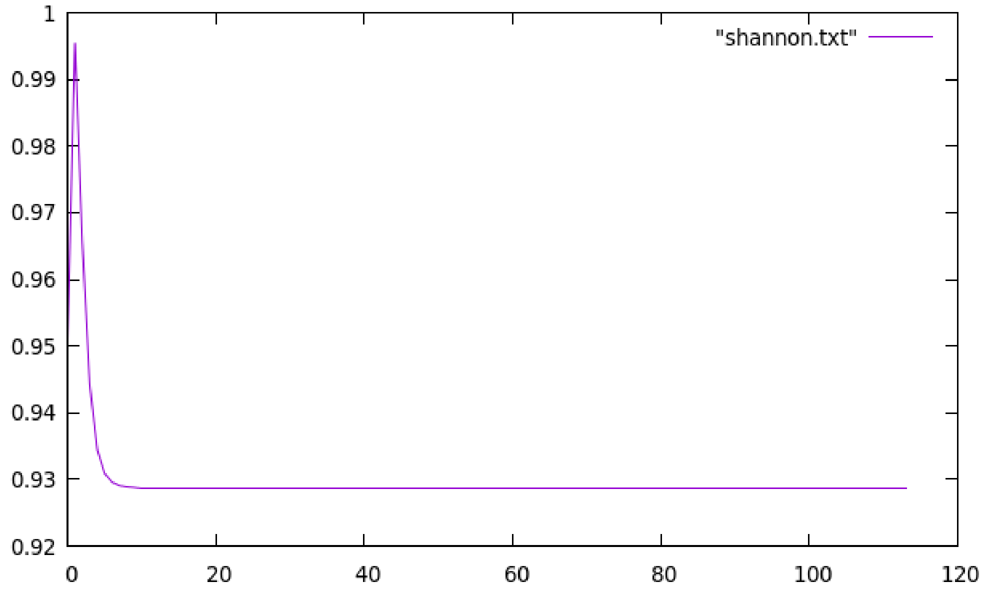

Figura 82: Shannon's Entropy

3.20. Observations: Rule B3/S12345678 at 37 %

From the three runs we can observe:

- The amount of live cells increases fast and even before reaching 20 iterations it stabilizes at almost 65 %.
- The Shannon's Entropy drops fast reaching 0.95 and stabilizes there.

3.21. Rule B234567/S456 at 50 %

Testing rule B234567/S456 with no pattern recognition activated, up to 10,000 iterations, 50 % live cells at the beginning.

3.21.1. 1st Run

System never stabilized. Variance was 6,549,084.7.

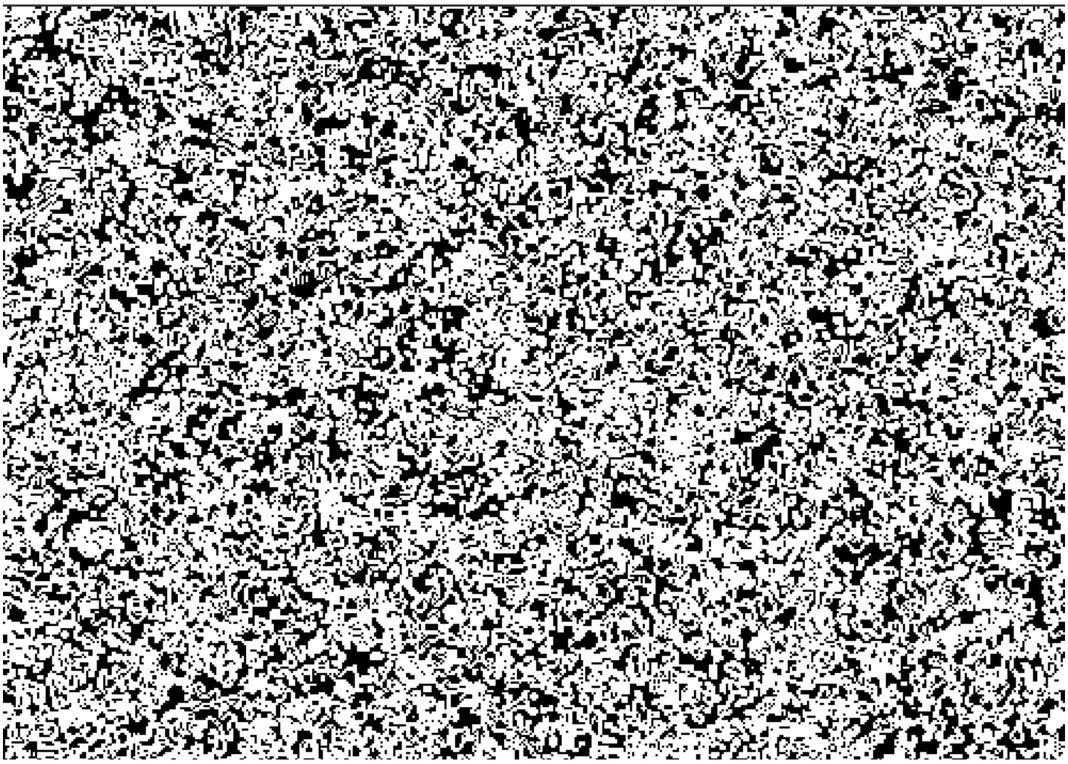

Figura 83: Simulation running

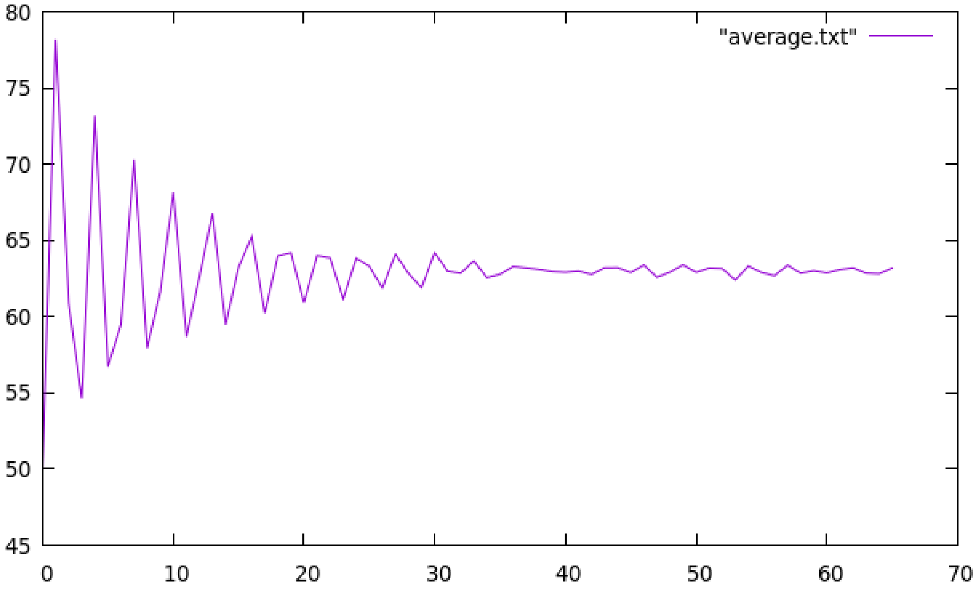

Figura 84: Average live cells

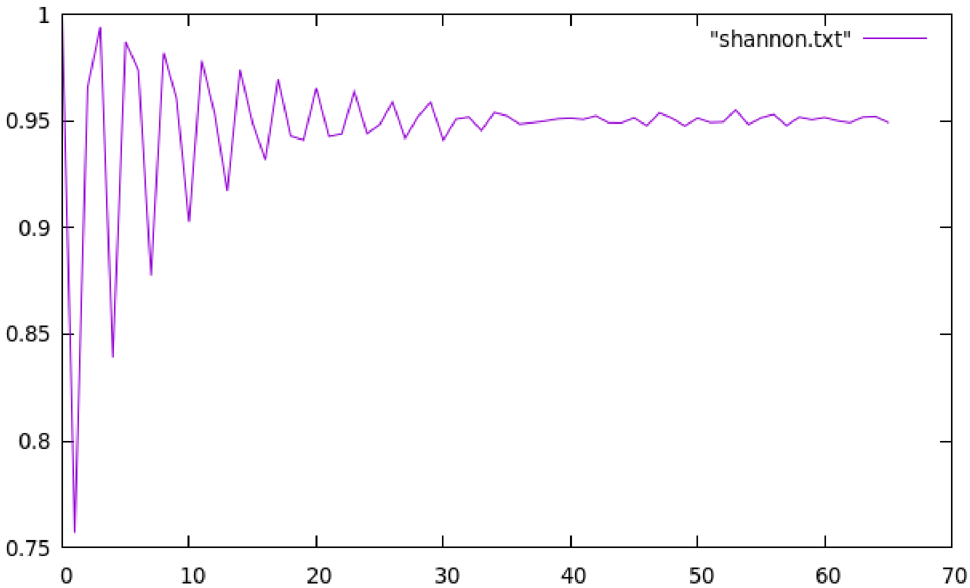

Figura 85: Shannon's Entropy

3.21.2. 2nd Run

System never stabilized. Variance was 6,351,496.6.

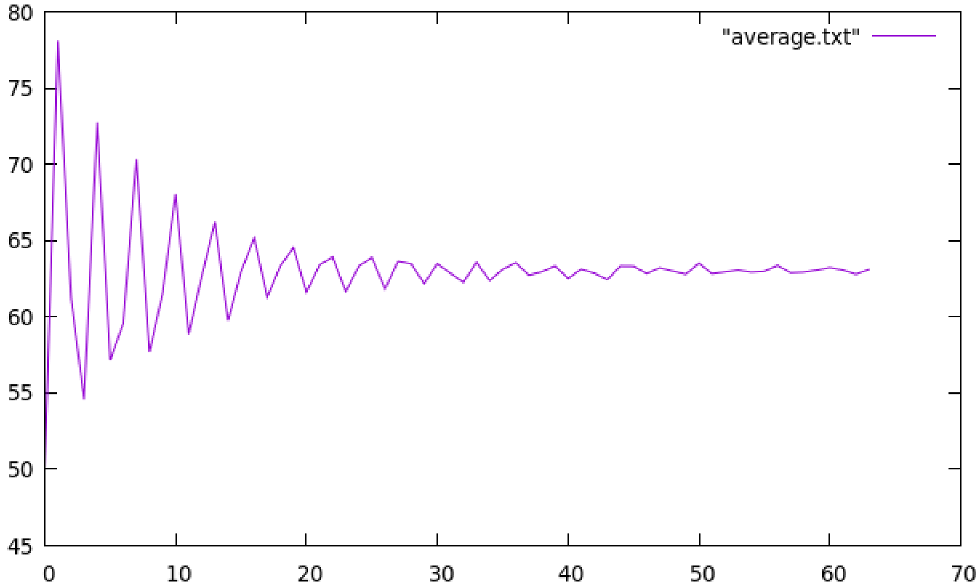

Figura 86: Average live cells

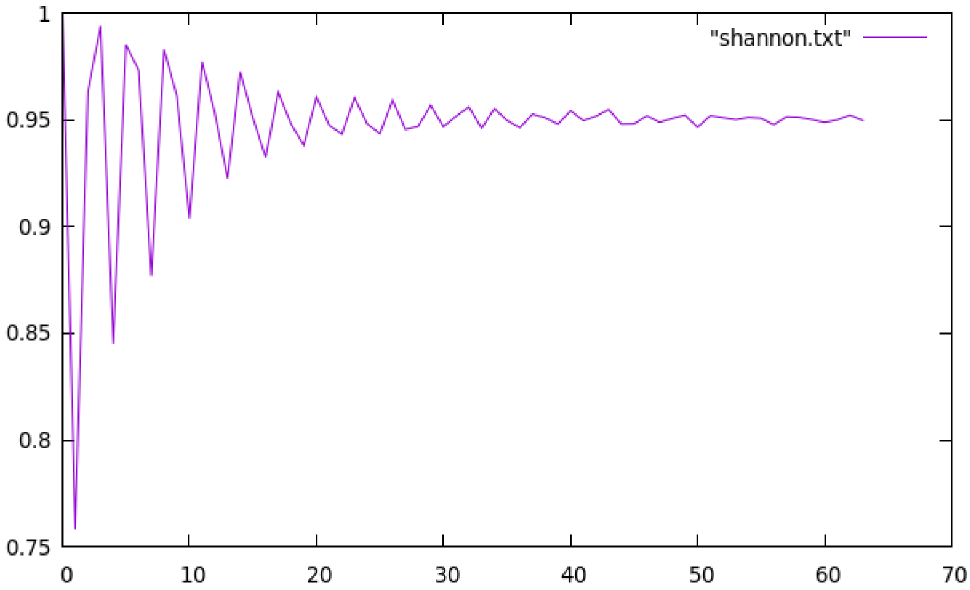

Figura 87: Shannon's Entropy

3.21.3. 3rd Run

System never stabilized. Variance was 6,357,545.8.

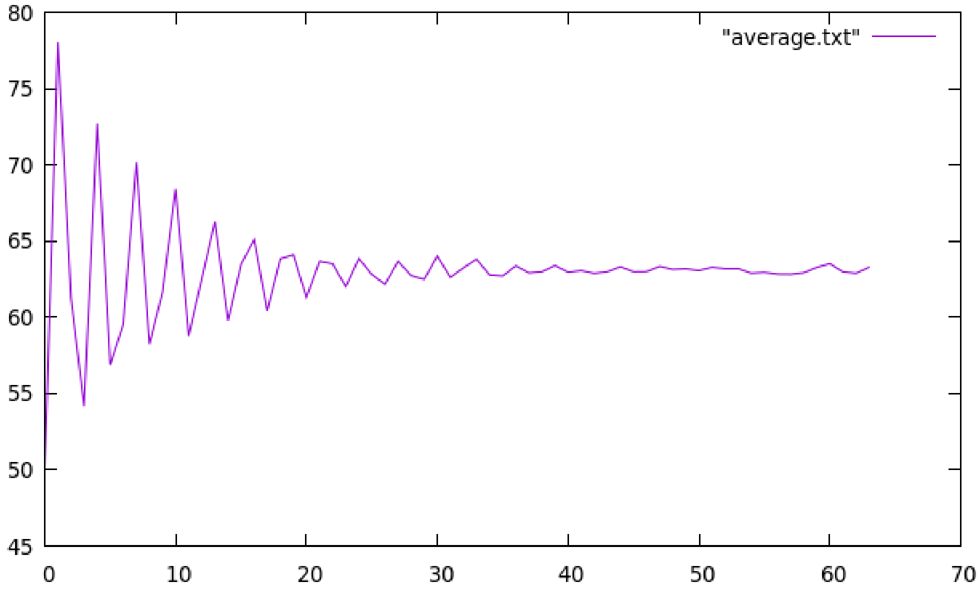

Figura 88: Average live cells

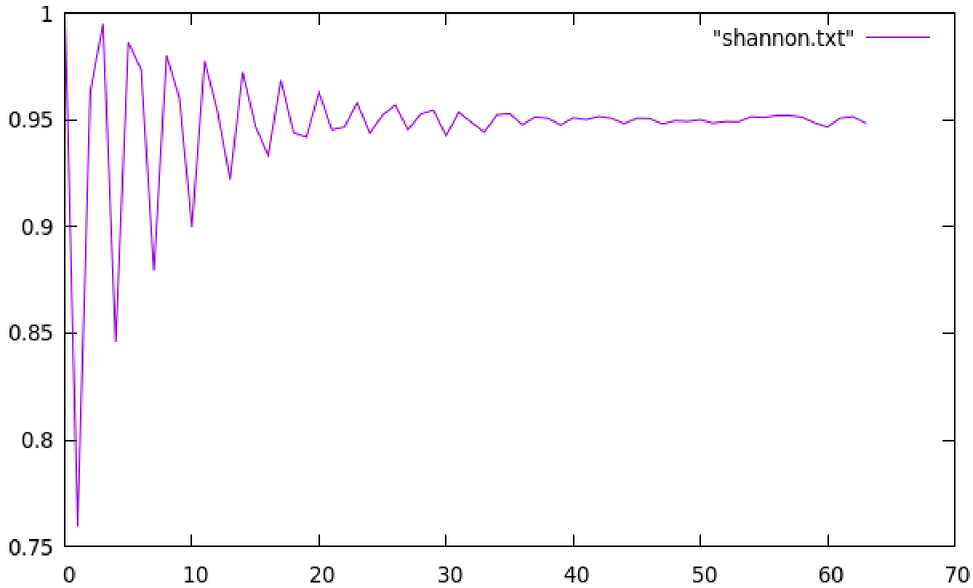

Figura 89: Shannon's Entropy

3.22. Observations: Rule B234567/S456 at 50 %

From the three runs we can observe:

- The percentage of live cells increases and decreases quickly in the first 30 iterations generating a riddle effect on the graph, then it stabilizes between 60 % and 65 % .
- The Shannon's Entropy behaves similarly to the average of live cells, generating a riddle effect on the graph and stabilizing at 0.95 near the 40 iteration.

3.23. Rule B234567/S456 at 37 %

Testing rule B234567/S456 with no pattern recognition activated, up to 10,000 iterations, 37 % live cells at the beginning.

3.23.1. 1st Run

System never stabilized. Variance was 6,306,452.7.

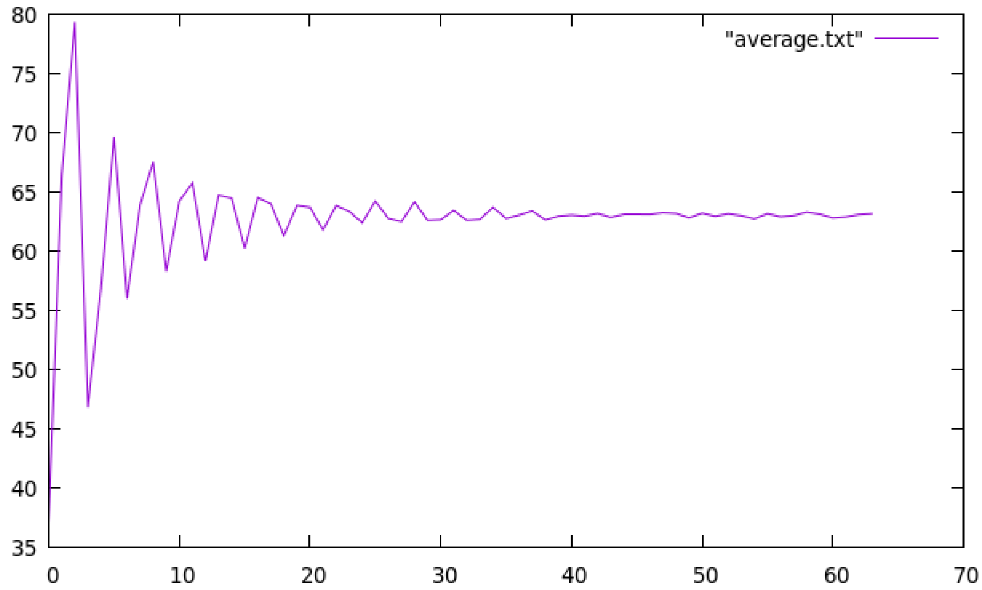

Figura 90: Average live cells

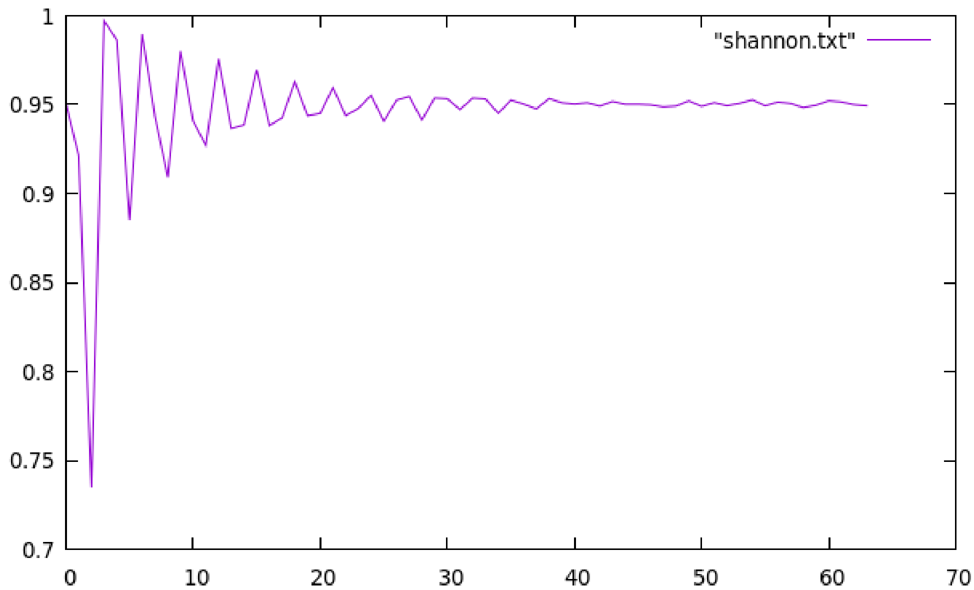

Figura 91: Shannon's Entropy

3.23.2. 2nd Run

System never stabilized. Variance was 6,213,146.2.

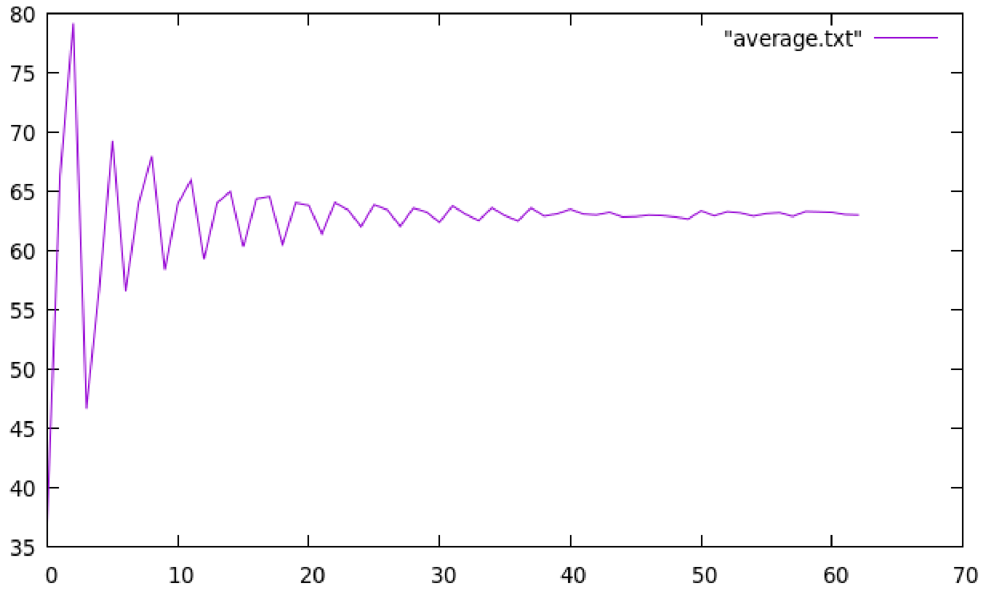

Figura 92: Average live cells

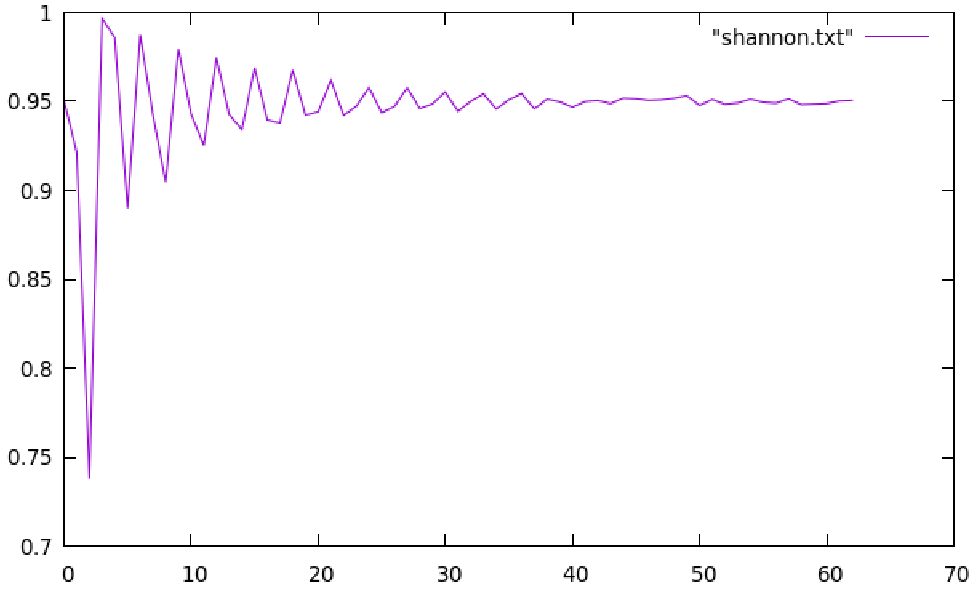

Figura 93: Shannon's Entropy

3.23.3. 3rd Run

System never stabilized. Variance was 6,207,177.

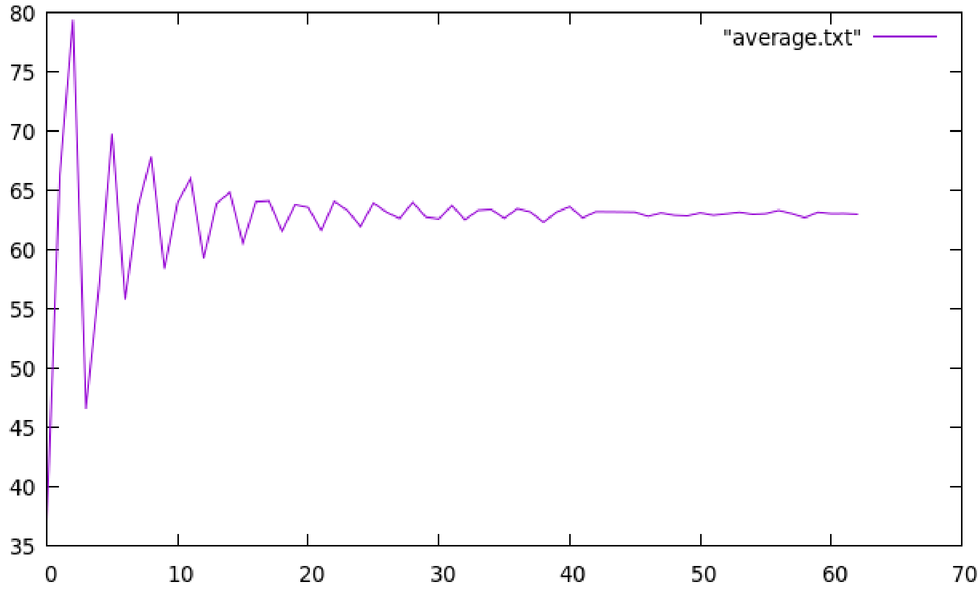

Figura 94: Average live cells

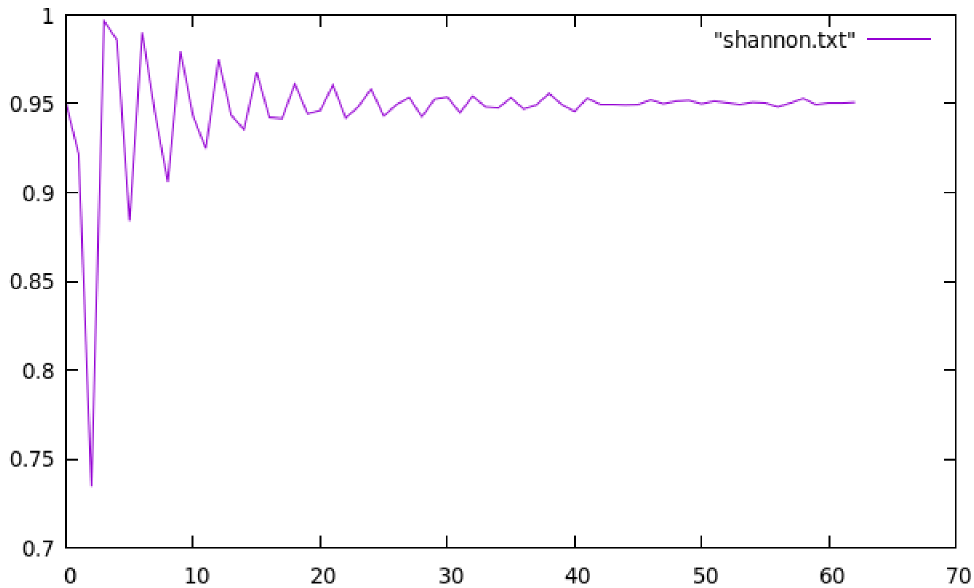

Figura 95: Shannon's Entropy

3.24. Observations: Rule B234567/S456 at 37 %

There was no significant difference between the configuration with 50 % and 37 %.

4. CODE

```
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

#include "olcConsoleGameEngineSDL.h"

class OneLoneCoder_GameOfLife : public olcConsoleGameEngine
{
public:
	OneLoneCoder_GameOfLife()
	{
		m_sAppName = L"Game_Of_Life";
	}

private:
	int *m_output, *m_state;
	long int nAlive, nDead, iteraciones = 0;
	int nNeighbours, nCells;
	double shannon = 0, probability_alive = 0, probability_dead = 0;
	double average, variance_sum = 0;
	ofstream avg_writer, shannon_writer, still_writer, oscilator_writer, chaos_writer;
	ifstream configuration_reader;
	int birth_rules[10], survive_rules[10];
	int birth_index, survive_index;
	bool survive, birth;
	long int i, j;
	//vector< stack<char> > stacks;
	int sumNeighbours, still_counter, oscilator_counter;
	bool nOne, nTwo, nThree, nFour, nSix, nSeven, nEight, nNine;
	string general_option;
	int pattern_switch;

protected:
	// Called by olcConsoleGameEngine
	virtual bool OnUserCreate()
	{
		nCells = ScreenWidth() * ScreenHeight();
		m_output = new int[ScreenWidth() * ScreenHeight()];
		m_state = new int[ScreenWidth() * ScreenHeight()];
		string numberOfOnesString, rules;
		string input_line, file_configuration_name;
		int numberOfOnesNumerator, numberOfOnesDenominator, counterOfOnes;
		long int random_index, line_offset, y;

		// CLEANING UP MEMORY SPACE
		memset(m_output, 0, ScreenWidth() * ScreenHeight() * sizeof(int));
		memset(m_state, 0, ScreenWidth() * ScreenHeight() * sizeof(int));

		// ADDING AND INITIALIZING A STACK FOR EVERY CELL
		/*
		for(i = 0; i < ScreenWidth() * ScreenHeight(); i++)
		{
			stack<char> aux;
			aux.push('Z');
			stacks.push_back(aux);
			// cout << aux.top() << endl;
		}
		*/

		// READING RULES
		cout << "Enter_rules:_";
		getline(cin, rules);
		birth_index = 0;
		survive_index = 0;
		if(rules[0] == 'B' || rules[0] == 'b')
		{
			i = 1; //Skipping 'B'
```

```
            while(rules[i] != '/')
                    birth_rules[birth_index++] = rules[i++] - '0';

            if(rules[i + 1] != 'S') // If there's no 'S'
            {
                    cout << "Error: Wrong syntax, should be an 'S' after '/'" << endl;
                    exit(EXIT_FAILURE);
            }
            i += 2; //Skipping '/' and 'S'

            while(i < rules.length())
                    survive_rules[survive_index++] = rules[i++] - '0';
    }
    else
    {
            cout << "Error: Wrong syntax" << endl;
            exit(EXIT_FAILURE);
    }

    // ASKING FOR INITIAL CONFIGURATION
    cout << "Do you want to load a configuration? Yes[Y-y] | No[N-n]: ";
    getline(cin, general_option);
    if(general_option[0] == 'Y' || general_option[0] == 'y')
    {
            // READING FILE INPUT NAME
            cout << "Enter the file name: ";
            getline(cin, file_configuration_name);
            ifstream configuration_reader(file_configuration_name);
            if(configuration_reader.is_open())
            {
                    // CUSTOM SET FUNCTION
                    auto set = [&](int x, int y, wstring s)
                    {
                            int p = 0;
                            for (auto c : s)
                            {
                                    m_state[y * ScreenWidth() + x + p] = c == L'#' ? 1
                                        : 0;
                                    p++;
                            }
                    };
                    // SETTING CONFIGURATION INTO M_STATE
                    y = 0;
                    while(getline(configuration_reader, input_line))
                    {
                            line_offset = 0;
                            for(auto c: input_line)
                            {
                                    m_state[y * ScreenWidth() + line_offset] = c == L'1
                                        ' ? 1 : 0;
                                    line_offset++;
                            }
                            y++;
                    }
                    configuration_reader.close();
            }
            else
            {
                    cout << "Error: Unable to open configuration file.";
                    exit(EXIT_FAILURE);
            }
    }
    else
    {
            // READING PERCENTAGE OF 1'S ACCURACY INCREMENTED V2
            cout << "Enter percentage of 1's: ";
            getline(cin, numberOfOnesString);
            if(numberOfOnesString[0] == '0') // i.e. 0.5%
            {
                    numberOfOnesNumerator = 0;
                    numberOfOnesDenominator = numberOfOnesString[2] - '0';
```

```
        numberOfOnesDenominator *= 10; // Applying scale i.e. 0.5% = 5(user
            input) * 10 = 50 of a total of 1000;
}
else if(numberOfOnesString.length() == 2) // i.e. 5%
{
        numberOfOnesString = numberOfOnesString.substr(0, 1);
        numberOfOnesNumerator = stoi(numberOfOnesString);
        numberOfOnesNumerator *= 10; // Applying scale i.e. 5% = 5(user
            input) * 10 = 50 of a total of 1000;
        numberOfOnesDenominator = 0;
}
else if(numberOfOnesString.length() == 3) // i.e. 50%
{
        numberOfOnesString = numberOfOnesString.substr(0, 2);
        numberOfOnesNumerator = stoi(numberOfOnesString);
        numberOfOnesNumerator *= 10; // Applying scale i.e. 50% = 50(user
            input) * 10 = 500 of a total of 1000;
        numberOfOnesDenominator = 0;
}
else // i.e. 70.5%
{
        numberOfOnesString = numberOfOnesString.substr(0, 2);
        numberOfOnesNumerator = stoi(numberOfOnesString);
        numberOfOnesNumerator *= 10; // Applying scale i.e. 70.5% = 70(user
            input) * 10 = 700 of a total of 1000;
        numberOfOnesDenominator = numberOfOnesString[3] - '0';
        numberOfOnesDenominator *= 10; // Applying scale i.e. 70.5% = 5(
            user input) * 10 = 50 of a total of 50;
}

// GENERATING FIRST M_STATE
counterOfOnes = 0;
for (i = 0; i < ScreenWidth() * ScreenHeight(); i++)
{
        if(((i + 1) % 1001) == 0)
        {
                while(counterOfOnes < (numberOfOnesNumerator +
                    numberOfOnesDenominator))
                {
                        //cout << "Counter of ones: " << counterOfOnes << "
                            < Number of ones: " << numberOfOnes << endl;
                        random_index = (i - 1000) + (rand() % 1000);
                        if(m_state[random_index] == 0)
                        {
                                m_state[random_index] = 1;
                                counterOfOnes++;
                        }
                }

                while(counterOfOnes > (numberOfOnesNumerator +
                    numberOfOnesDenominator))
                {
                        //cout << "Counter of ones: " << counterOfOnes << "
                            > Number of ones: " << numberOfOnes << endl;
                        random_index = (i - 1000) + (rand() % 1000);
                        if(m_state[random_index] == 1)
                        {
                                m_state[random_index] = 0;
                                counterOfOnes--;
                        }
                }
                counterOfOnes = 0;
        }

        if((rand() % 2) == 1)
        {
                counterOfOnes++;
                m_state[i] = 1;
        }
        else
                m_state[i] = 0;
}
```

```
}

// ASKING FOR PATTERN RECOGNITION
cout << "Pattern Recognition -> 1-[Osc-Still-Chaos] | 2-[Static-NonStatic] | 3-[
    NoRecognition] : ";
getline(cin, general_option);
if(general_option[0] == '1')
        pattern_switch = 1;
else if(general_option[0] == '2')
        pattern_switch = 2;
else
        pattern_switch = 3;

/*
//Diehard
set(80, 50, L"       #");
set(80, 51, L"##  ");
set(80, 52, L" #   ###");
*/

/*
//Acorn
set(80, 50, L" #");
set(80, 51, L"   #");
set(80, 52, L"##  ###");
*/

/*
// R-Pentomino
set(80, 50, L" ## ");
set(80, 51, L"##  ");
set(80, 52, L" #  ");
*/


// Gosper Glider Gun
// set(60, 45, L"........................#............");
// set(60, 46, L"......................#.#............");
// set(60, 47, L"............##......##............##.");
// set(60, 48, L"...........#...#....##............##.");
// set(60, 49, L"##........#.....#...##...............");
// set(60, 50, L"##........#...#.##....#.#............");
// set(60, 51, L"..........#.....#.......#............");
// set(60, 52, L"...........#...#.....................");
// set(60, 53, L"............##.......................");


/*
// ORIGIN
set(60, 53, L"#..");
set(60, 54, L"..#");
set(60, 55, L"..#");
set(60, 56, L"#..");

set(84, 53, L"..#");
set(84, 54, L"#..");
set(84, 55, L"#..");
set(84, 56, L"..#");
*/

// Infinite Growth
//set(20, 50, L"########.#####...###......#######.#####");

//OPENING FILES
avg_writer.open("average.txt", ios_base::out | ios_base::trunc);
if(!avg_writer)
{
        cout << "Error: Cannot open 'average' file" << endl;
        return true;
}
shannon_writer.open("shannon.txt", ios_base::out | ios_base::trunc);
if(!shannon_writer)
```

```
        {
                cout << "Error: Cannot open 'shannon' file" << endl;
                return true;
        }
        still_writer.open("still.txt", ios_base::out | ios_base::trunc);
        if(!still_writer)
        {
                cout << "Error: Cannot open 'still' file" << endl;
                return true;
        }
        oscilator_writer.open("oscilator.txt", ios_base::out | ios_base::trunc);
        if(!oscilator_writer)
        {
                cout << "Error: Cannot open 'oscilator' file" << endl;
                return true;
        }
        chaos_writer.open("chaos.txt", ios_base::out | ios_base::trunc);
        if(!chaos_writer)
        {
                cout << "Error: Cannot open 'chaos' file" << endl;
                return true;
        }
        // SETTING PRECISION
        cout.precision(8);
        avg_writer.precision(8);
        shannon_writer.precision(8);
        still_writer.precision(8);
        oscilator_writer.precision(8);
        chaos_writer.precision(8);

        return true;
}



// Called by olcConsoleGameEngine
virtual bool OnUserUpdate(float fElapsedTime)
{
        // HANDLING USER INPUT
        if(m_keys[VK_ESCAPE].bPressed)
        {
                cout << "Do you want to save the configuration? Yes[Y-y] | No[N-n]: ";
                getline(cin, general_option);
                // SAVING CONFIGURATION
                if(general_option[0] == 'Y' || general_option[0] == 'y')
                {
                        ofstream configuration_writer("saved.txt");
                        if(configuration_writer.is_open())
                        {
                                for(i = 0; i < ScreenWidth() * ScreenHeight(); i++)
                                {
                                        if(!(i % ScreenWidth()) && i != 0)
                                                configuration_writer << endl;
                                        if(m_output[i])
                                                configuration_writer << '1';
                                        else
                                                configuration_writer << '0';
                                }
                        }
                        else
                                cout << "Error: Unable to open configuration writer" <<
                                    endl;
                        configuration_writer.close();
                }
                // WRITTING VARIANCE
                ofstream variance_writer("variance.txt");
                if(variance_writer.is_open())
                {
                        variance_writer.precision(8);
                        variance_writer << (double) variance_sum / nCells << endl;
                }
                else
```

```
                cout << "Error: Unable to open variance writer" << endl;
            // CLOSING ALL WRITERS
            variance_writer.close();
            avg_writer.close();
            shannon_writer.close();
            still_writer.close(); // What is green
            oscilator_writer.close(); // What is yellow
            chaos_writer.close(); // What is white
            exit(EXIT_SUCCESS);
    }
    if(m_keys[VK_SPACE].bHeld)
            return true;
    else if(m_keys[VK_F1].bHeld)
            this_thread::sleep_for(100ms);
    else if(m_keys[VK_F2].bHeld)
            this_thread::sleep_for(200ms);
    else if(m_keys[VK_F3].bHeld)
            this_thread::sleep_for(300ms);
    else if(m_keys[VK_F5].bHeld)
            this_thread::sleep_for(500ms);
    else if(m_keys[VK_F10].bHeld)
            this_thread::sleep_for(1000ms);
    else if(m_keys[VK_F11].bHeld)
            this_thread::sleep_for(2000ms);

    // ITERATIONS STOP
    if(iteraciones > 10000)
            return true;
    // TWO HANDLY FUNCTIONS
    auto cell = [&](int x, int y)
    {
            return m_output[y * ScreenWidth() + x];
    };
    auto cellt1 = [&](int x, int y)
    {
            return m_state[y * ScreenWidth() + x];
    };
    // STORE OUTPUT STATE
    for (i = 0; i < ScreenWidth() * ScreenHeight(); i++)
            m_output[i] = m_state[i];

    //
    nAlive = 0;
    still_counter = 0;
    oscilator_counter = 0;
    for (int x = 0; x < ScreenWidth(); x++)
            for (int y = 0; y < ScreenHeight(); y++)
            {
                    nNeighbours = 0;

                    // COUNTING NEIGHBOURS AND THEIR POSITIONS REQUIRED WHEN PATTERN
                        RECOGNITION OSC-STILL-CHAOS IS ON
                    if(pattern_switch == 1)
                    {
                            nOne = false;
                            nTwo = false;
                            nThree = false;
                            nFour = false;
                            nSix = false;
                            nSeven = false;
                            nEight = false;
                            nNine = false;

                            if(cell((x - 1 + ScreenWidth()) %ScreenWidth(), (y - 1 +
                                ScreenHeight()) %ScreenHeight()) == 1)
                            {
                                    nNeighbours++;
                                    nOne = true;
                            }
                            if(cell((x - 0 + ScreenWidth()) %ScreenWidth(), (y - 1 +
                                ScreenHeight()) %ScreenHeight()) == 1)
                            {
```

```
                        nNeighbours++;
                        nTwo = true;
                }
                if(cell((x + 1 + ScreenWidth()) % ScreenWidth(), (y - 1 +
                    ScreenHeight()) % ScreenHeight()) == 1)
                {
                        nNeighbours++;
                        nThree = true;
                }
                if(cell((x - 1 + ScreenWidth()) % ScreenWidth(), (y - 0 +
                    ScreenHeight()) % ScreenHeight()) == 1)
                {
                        nNeighbours++;
                        nFour = true;
                }
                if(cell((x + 1 + ScreenWidth()) % ScreenWidth(), (y - 0 +
                    ScreenHeight()) % ScreenHeight()) == 1)
                {
                        nNeighbours++;
                        nSix = true;
                }
                if(cell((x - 1 + ScreenWidth()) % ScreenWidth(), (y + 1 +
                    ScreenHeight()) % ScreenHeight()) == 1)
                {
                        nNeighbours++;
                        nSeven = true;
                }
                if(cell((x - 0 + ScreenWidth()) % ScreenWidth(), (y + 1 +
                    ScreenHeight()) % ScreenHeight()) == 1)
                {
                        nNeighbours++;
                        nEight = true;
                }
                if(cell((x + 1 + ScreenWidth()) % ScreenWidth(), (y + 1 +
                    ScreenHeight()) % ScreenHeight()) == 1)
                {
                        nNeighbours++;
                        nNine = true;
                }
        }
        else
        {
                // NORMAL WAY OF COUNTING NEIGHBOURS
                for(i = -1; i < 2; i++)
                        for(j = -1; j < 2; j++)
                                nNeighbours += cell((x + i + ScreenWidth())
                                     % ScreenWidth(), (y + j + ScreenHeight
                                    ()) % ScreenHeight());

                nNeighbours -= cell(x, y);
        }

        // APPLYING SURVIVE RULES
        if(cell(x, y) == 1)
        {
                survive = false;
                for(i = 0; i < survive_index; i++)
                        if(nNeighbours == survive_rules[i])
                        {
                                m_state[y * ScreenWidth() + x] = 1;
                                survive = true;
                                break;
                        }

                if(!survive)
                        m_state[y * ScreenWidth() + x] = 0;
        }
        else // APPLYING BIRTH RULES
        {
                birth = false;
                for(int i = 0; i < birth_index; i++)
                        if(nNeighbours == birth_rules[i])
```

```
                        {
                                m_state[y * ScreenWidth() + x] = 1;
                                birth = true;
                                break;
                        }

                if(!birth)
                        m_state[y * ScreenWidth() + x] = 0;
        }

        // PATTERN RECOGNITION PROCESS
        if(pattern_switch == 1) // Handle Oscilators-Still-Gliders-Chaos
        {
                // AUTOMATA NEIGHBOURS POSITIONS V1
                if(cell(x, y) == 1)
                {
                        nAlive++;
                        if(cellt1(x, y))
                        {
                                still_counter++;
                                //Block
                                if(nSix && nEight && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nFour && nSeven && nEight)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nTwo && nThree && nSix)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nOne && nTwo && nFour)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                //Bee-hive
                                else if(nThree && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nSix && nSeven)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nFour && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nOne && nSeven)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nOne && nSix)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nThree && nFour)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nThree && nEight)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nTwo && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                // Loaf
                                else if(nThree && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nSix && nSeven)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nFour && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nOne && nNine)
                                        Draw(x, y, PIXEL_SOLID,
                                            FG_DARK_GREEN);
                                else if(nOne && nThree)
```

```
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nOne && nEight)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nTwo && nSeven)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        // Boat
        else if(nSix && nEight)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nFour && nSeven && nNine)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nTwo && nThree && nSeven)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nOne && nSeven)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nOne && nThree)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nSix && nSeven && nNine)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nTwo && nFour)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        // Tub
        else if(nSeven && nNine)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nThree && nNine)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nOne && nSeven)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        else if(nOne && nThree)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        // Steady
        else if(nOne && nSeven && nNine)
                Draw(x, y, PIXEL_SOLID,
                    FG_DARK_GREEN);
        // Blinker
        else if(nFour && nSix)
        {
                Draw(x, y, PIXEL_SOLID, FG_YELLOW);
                still_counter--;
                oscilator_counter++;
        }
        else if(nTwo && nEight)
        {
                Draw(x, y, PIXEL_SOLID, FG_YELLOW);
                still_counter--;
                oscilator_counter++;
        }
        else
        {
                Draw(x, y, PIXEL_SOLID, FG_WHITE);
                still_counter--;
        }
}
else
{
        oscilator_counter++;
        if(nSix && m_state[y * ScreenWidth() + x]
            == 0)
                Draw(x, y, PIXEL_SOLID, FG_YELLOW);
```

```
                            else if(nFour && m_state[y * ScreenWidth()
                                + x] == 0)
                                    Draw(x, y, PIXEL_SOLID, FG_YELLOW);
                            else if(nEight && m_state[y * ScreenWidth()
                                 + x] == 0)
                                    Draw(x, y, PIXEL_SOLID, FG_YELLOW);
                            else if(nTwo && m_state[y * ScreenWidth() +
                                 x] == 0)
                                    Draw(x, y, PIXEL_SOLID, FG_YELLOW);
                            else
                            {
                                    Draw(x, y, PIXEL_SOLID, FG_WHITE);
                                    oscilator_counter--;
                            }
                    }


//              // Glider Fase 1
//              else if(nNine)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nOne && nThree && nSix)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nTwo && nFour)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nTwo && nSeven && nEight)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nEight)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              // Glider Fase 2
//              else if(nThree && nSix)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nTwo && nThree && nFour)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nOne && nSix && nSeven && nEight)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nFour && nSeven)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              // Glider Fase 3
//              else if(nSix)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nThree && nFour && nSix)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nTwo && nFour)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nOne && nSeven && nEight)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              // Glider Fase 4
//              else if(nSeven && nEight)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nOne && nThree && nSix && nEight)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nTwo && nFour && nSeven)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(nTwo && nThree)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_RED);
//              else if(m_state[y * ScreenWidth() + x] == 1)
//                      Draw(x, y, PIXEL_SOLID, FG_DARK_GREEN);
//              else
//                      Draw(x, y, PIXEL_SOLID, FG_WHITE);
            }
            else
                    Draw(x, y, PIXEL_SOLID, FG_BLACK);
}
else if(pattern_switch == 2) // Hanldes Static and Non Static
    Pattern
{
        // DRAWS STATIC AND NON STATIC CELLS
        if (cell(x, y) == 1)
        {
                if(m_state[y * ScreenWidth() + x] == 1)
                {
                        Draw(x, y, PIXEL_SOLID, FG_DARK_GREEN);
```

```
                                                still_counter++;
                                        }
                                        else
                                        {
                                                Draw(x, y, PIXEL_SOLID, FG_WHITE);
                                                oscilator_counter++;
                                        }
                                                nAlive++;
                                }
                                else
                                        Draw(x, y, PIXEL_SOLID, FG_BLACK);
                        }
                        else
                        {
                                // DRAWS CELLS OLD FASHION
                                if (cell(x, y) == 1)
                                {
                                        Draw(x, y, PIXEL_SOLID, FG_WHITE);
                                        nAlive++;
                                }
                                else
                                        Draw(x, y, PIXEL_SOLID, FG_BLACK);
                        }
                }

                // CALCULUS SHANNON
                nDead = ScreenHeight() * ScreenWidth() - nAlive;
                probability_alive = (double) nAlive / (ScreenWidth() * ScreenHeight());
                probability_dead = (double) nDead / (ScreenWidth() * ScreenHeight());
                shannon = - probability_alive * log2(probability_alive) - probability_dead * log2(
                    probability_dead);
                // CALCULUS VARIANCE
                average = ((double) nAlive / nCells) * 100;
                variance_sum += pow((double) nAlive - average, 2);

                // WRITTING
                avg_writer << average << endl;
                shannon_writer << shannon << endl;
                shannon = 0;

                if(pattern_switch == 1 || pattern_switch == 2)
                {
                        still_writer << ((double) still_counter / nCells) * 100 << endl;
                        oscilator_writer << ((double) oscilator_counter / nCells) * 100 << endl;
                        chaos_writer << ((double) (nCells - nDead - still_counter -
                            oscilator_counter) / nCells) * 100 << endl;
                }

                cout << iteraciones++ << endl;

                return true;
        }
};


int main()
{
        // Seed random number generator
        srand(clock());

        // Use olcConsoleGameEngine derived app
        OneLoneCoder_GameOfLife game;
        game.ConstructConsole(500, 500, 2, 2);
        game.Start();

        return 0;
}
```