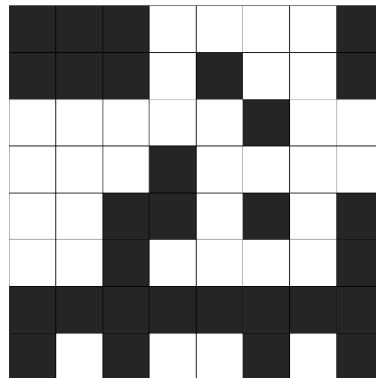




The University of Western Australia
School of Computer Science & Software Engineering
CITS3402: High Performance Computing
Semester 2, 2018

Assignment 1 Report:

Conway's Game of Life with OpenMP



Group Members:

Terence Fuhr (Student No. 10319759)

Juncai Liu (Student No. 21995495)

Issued: 15th September 2018

Table of Contents

Compilation Instructions	3
Running Instructions	3
Results	3
Timings	3
Plots	7
8x8 Grid	7
128 x 128 Grid	8
256 x 256 Grid	9
512 x 512 Grid	11
1024 x 1024 Grid	12
2048 x 2048 Grid	14

Compilation Instructions

1. Navigate in the terminal / command line interface to the directory that you have saved our assign1.c file
2. Enter the following:
 - a. Linux: `prompt> gcc -fopenmp -o assign1 assign1.c`
 - b. Windows: `prompt> gcc -fopenmp assign1.c -o assign1.exe`

Running Instructions

To make the program flexible for the marker/user, we have created the following command line arguments to customise how the program is run:

Linux Bash Prompt> `./assign1 rows cols steps output [threads]`
 Windows CMD Prompt> `assign1.exe rows cols steps output [threads]`

where:

argv[1] **rows** : the number of rows in the Game of Life board
 argv[2] **cols** : the number of columns in the Game of Life board
 argv[3] **steps** : the number of time steps to simulate
 argv[4] **output** : whether to output game state for each step to output.txt
 (t = true, f = false)
 not recommended for rows x cols x steps > 1,000,000
 argv[5] **threads** : (optional) The number of threads to run for the parallel region.
 If not entered, will default to default set by OS.
 Entering '1' effectively runs a purely sequential program.

Example usage: `./assign1 2048 2048 100 f 4`

Results

Timings

Time trials were conducted using a laptop with the following CPU specifications:

- Cores: 2
- Threads: 4 i.e. Hyperthreading (HT) available
- Clock speed: 2.6 GHz
- Architecture: 64bit

The table below shows the results of the trials for our program. Tests were conducted with 1 thread (sequential), 2 threads (parallel with hyperthreading off) and 4 threads (parallel with hyperthreading on). Three trials were conducted for each grid size, with the initial grid pattern changing every time due to the randomisation of the initial grid in the code.

To get a true comparison, the timer start and end points in the C program were just before and after the parallel region of code. To reduce overheads, outputting of game states to output.txt was turned off for the time trials, i.e. the run arguments were

```
./assign1 <rows> <cols> 100 f <threads>
```

Grid Size	Total Iterations	Trial No.	Run Time (sec)			Seq. : Par. Ratio (HT Off)	Seq. : Par. Ratio (HT On)
			Sequential	Parallel (HT Off)	Parallel (HT On)		
128 x 128	1,638,400	1	0.0312	0.0312	0.0312	1.000	1.000
		2	0.0313	0.0312	0.0312	1.002	1.002
		3	0.0312	0.0279	0.0279	1.119	1.119
		Mean	0.0312	0.0301	0.0301	1.037	1.037
256 x 256	6,553,600	1	0.1249	0.1250	0.0625	1.000	2.000
		2	0.1406	0.1250	0.0513	1.125	2.740
		3	0.1250	0.1250	0.0625	1.000	2.001
		Mean	0.1302	0.1250	0.0588	1.042	2.216
512 x 512	26,214,400	1	0.4971	0.4530	0.2656	1.097	1.872
		2	0.6429	0.4218	0.2663	1.524	2.415
		3	0.6294	0.4366	0.2911	1.442	2.162
		Mean	0.5898	0.4372	0.2743	1.349	2.150
1024 x 1024	104,857,600	1	2.0155	1.3996	0.8979	1.440	2.245
		2	1.9958	1.2119	0.9009	1.647	2.215
		3	2.0147	1.1903	0.9011	1.693	2.236
		Mean	2.0087	1.2673	0.9000	1.585	2.232
2048 x 2048	419,430,400	1	10.7498	6.6135	4.8949	1.625	2.196
		2	10.7002	6.4232	3.6028	1.666	2.970
		3	11.0620	6.4555	3.6220	1.714	3.054
		Mean	10.8373	6.4974	4.0399	1.668	2.683

From the table, the run time doesn't vary much from trial to trial for the same grid size and mode (sequential, parallel with threading on/off). This is to be expected by examining the main loop that simulates the time steps of the game:

```
for (int s = 0; s < steps; s++) {
    #pragma omp parallel for num_threads(threads)
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int neighbours = countneighbours(matrix1, i, j); // Count live neighbours

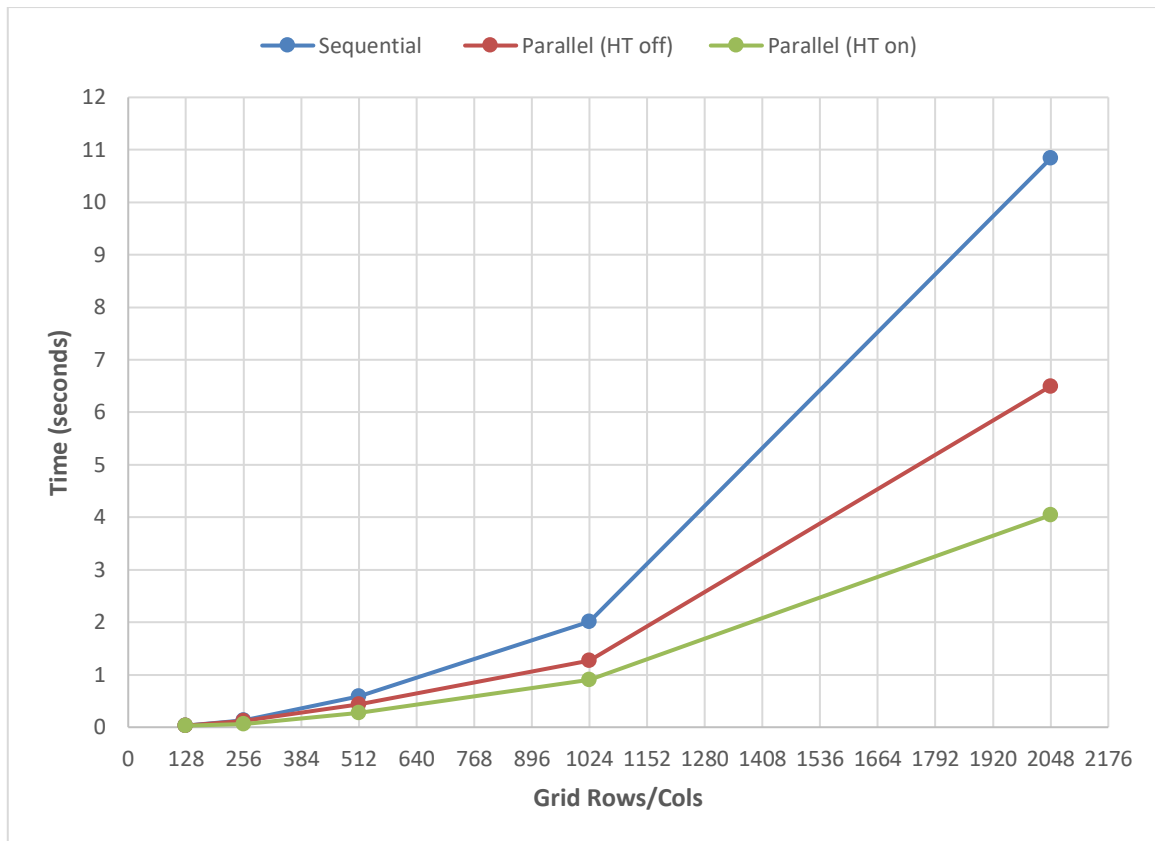
            if (matrix1[i][j] == 1) { // if the given cell is alive:
                if (neighbours == 2 || neighbours == 3) {
                    matrix2[i][j] = 1; // cell stays alive
                } else { // i.e. if neighbours = 0, 1, or 4
```

```
        matrix2[i][j] = 0; // cell dies
    }
} else { // i.e. else if the given cell is dead
    if (neighbours == 3) {
        matrix2[i][j] = 1; // cell becomes alive
    } else { // i.e. if neighbours = 0, 1, 2 or 4
        matrix2[i][j] = 0; // cell remains dead
    }
}
}
} // end of parallel region
```

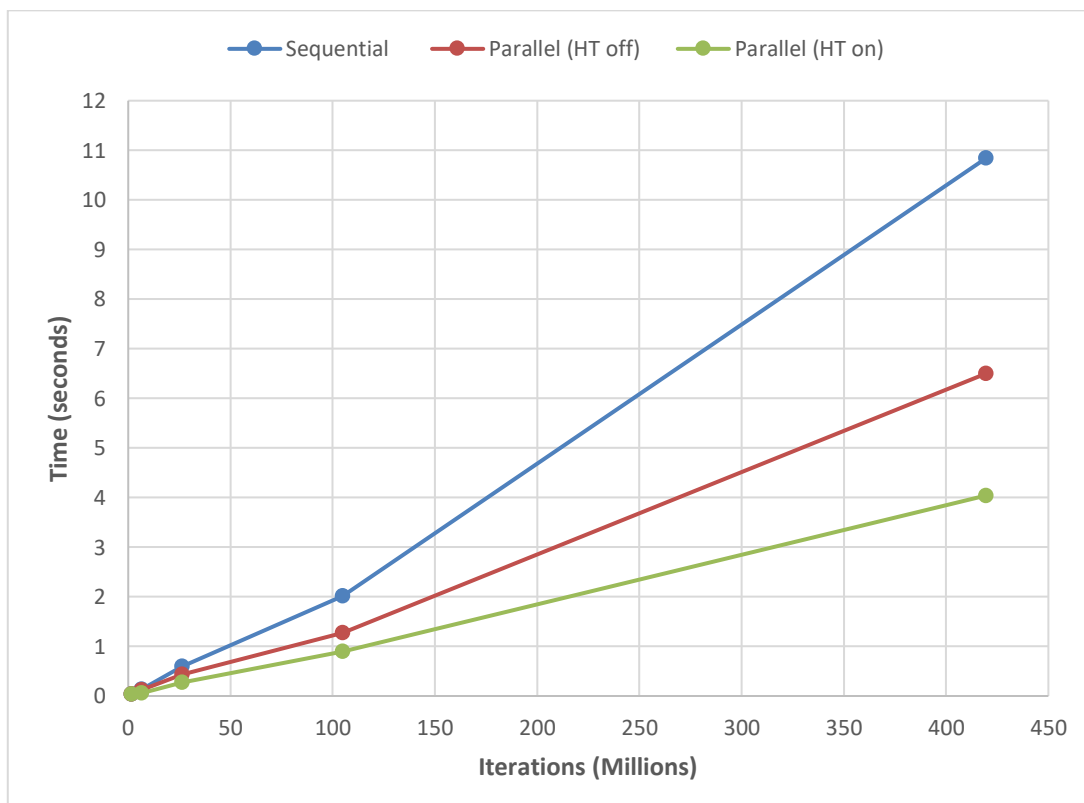
From the above code, this main loop always performs time steps * rows * columns iterations, which of course will be the same for each of these 3 runs, irrespective of the initial pattern. Furthermore, the 2-level nested if-else block ensures that for each cell/iteration there are always exactly 2 Boolean expressions to evaluate in all cases, leading to consistent run-times irrespective of the initial pattern. The time variations run to run are in fact more likely due to fluctuations in background activity/processes of the test computer.

We also observe that the ratio of time for sequential versus 2-thread parallel (hyperthreading off) is close to 1 for small grid sizes, indicating as expected that due to the overheads in managing parallel threading, the time savings at this size are small. As the grid size increases, the ratio tends towards the theoretical optimal of 2 for the 2 cores. For hyperthreading, we see that although the ratio exceeds 2 and reaches 2.7, it doesn't tend to 4, reflecting that hyperthreading on a dual core CPU doesn't perform quite as well as a quad core CPU with no hyperthreading.

Plotting the average times of the 3 trials for the various grid sizes and run modes are shown below. The increase in efficiency of parallelisation is shown by the greater divergence in the ratios of times as the grid size increases.



The run time is approximately parabolic with respect to the side length (no. of rows or no. columns) of the grid, which is to be expected given that the number of iterations is proportional to the no. rows 2 . In fact, below is a plot of the average run times against the total number of iterations, which is nearly linear as expected.

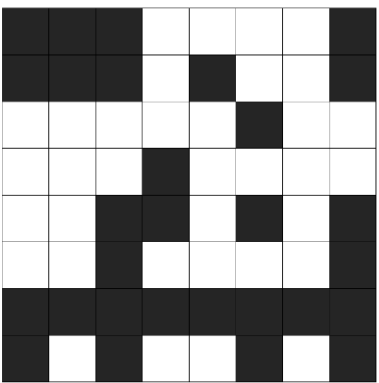
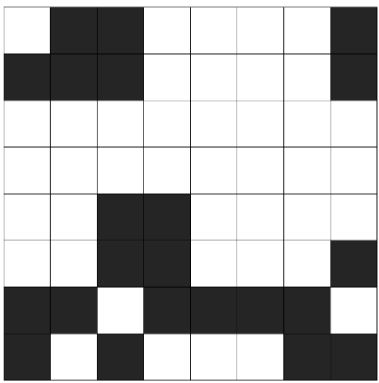
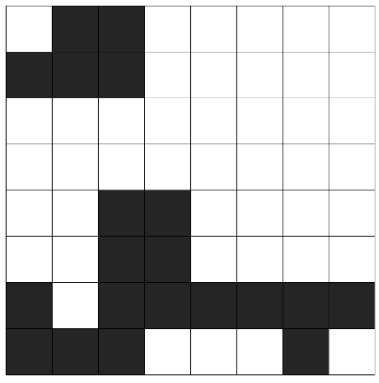


Plots

The plots of the Game of Life grids at various time steps are shown below.

8x8 Grid

To clearly illustrate the correctness of our program, below is a small 8x8 grid plotted using a custom script we created in Python to visualise the output in the output.txt file. Live cells are black and dead ones are white.

Step 0 (initial)	Step 1	Step 2
		

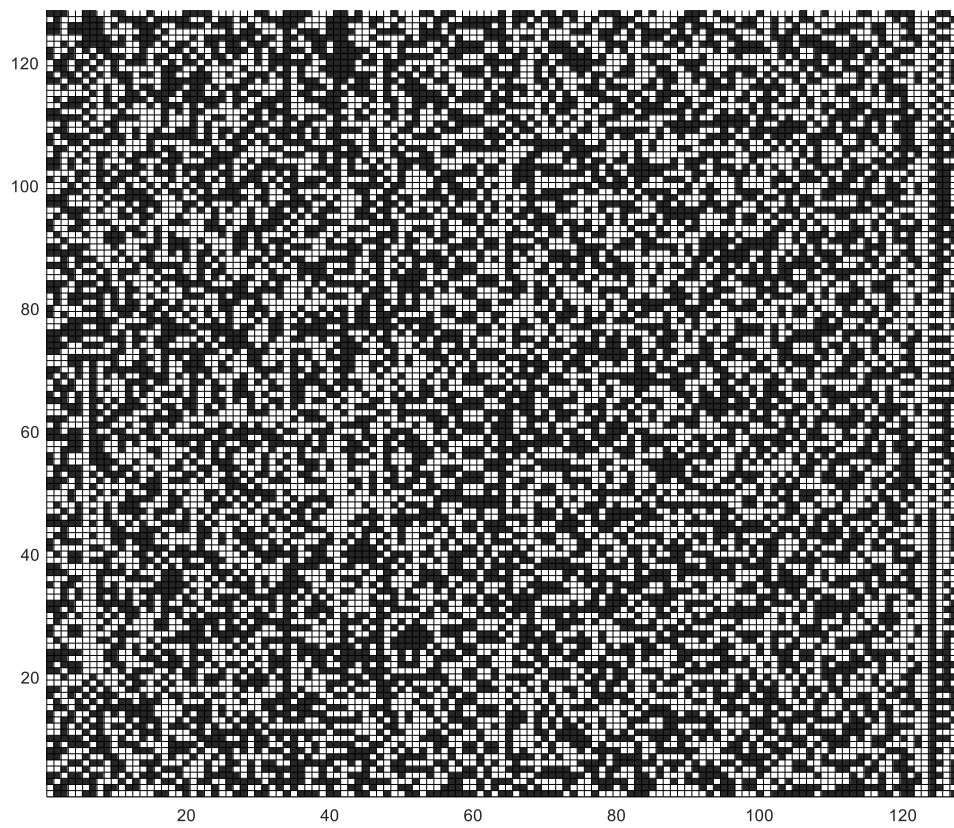
We found that generally the live cell pattern stabilised quite quickly (less than 10 steps) for the smaller grid sizes to a stable set of "still life" formations. It appears that using only 4 adjacent cells - rather than the traditional 8 - to compute the life/death of cell leads to a faster stabilisation.

For larger plots we used MATLAB, because our Python program only rendered plots to the screen, rather than to an image file, meaning that our plots were limited by our screen resolution, which was insufficient for larger grid sizes. Rather than outputting the game state at every step (enabled by setting the command line argument 'output' to 't'), we modified our code to only output the game state at time steps 0 (initial), 1, and 100 (last) for plotting in Matlab.

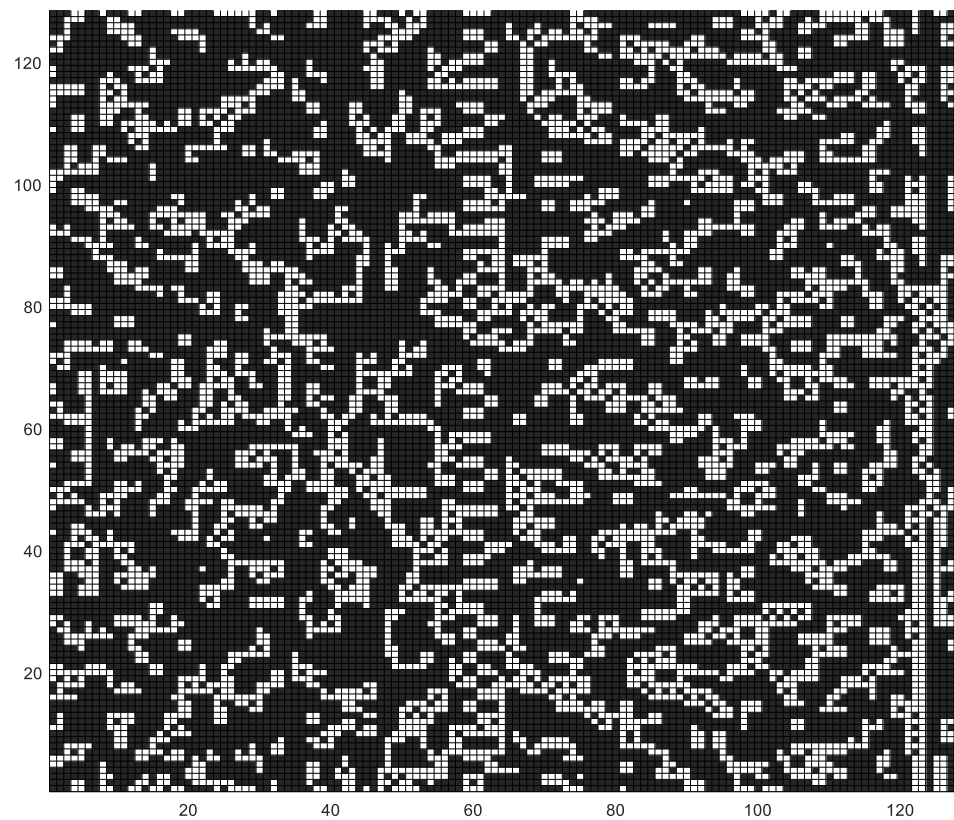
Sample code for plotting in Matlab:

```
f = dlmread('output0.txt');
img = imagesc('CData',f);
hold on;
size = 128;
for i = 1:size % draw gridlines to separate the cells clearly
    plot([.5,size+.5],[i-.5,i-.5],'k-');
    plot([i-.5,i-.5],[.5,size+.5],'k-');
end
```

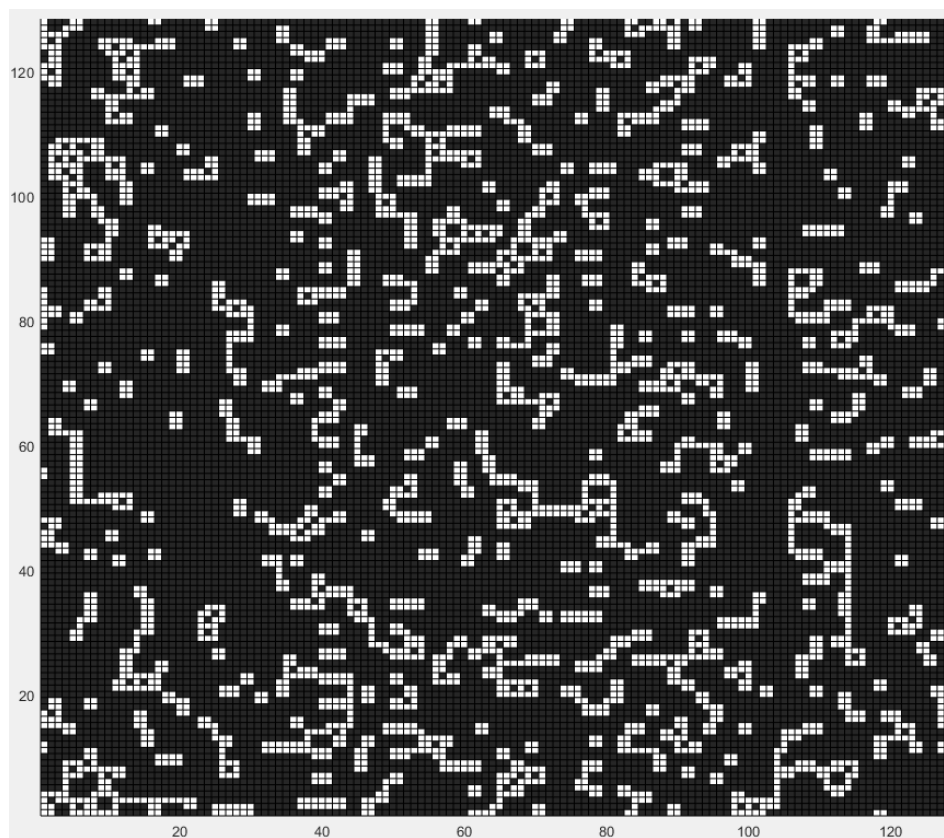
128 x 128 Grid



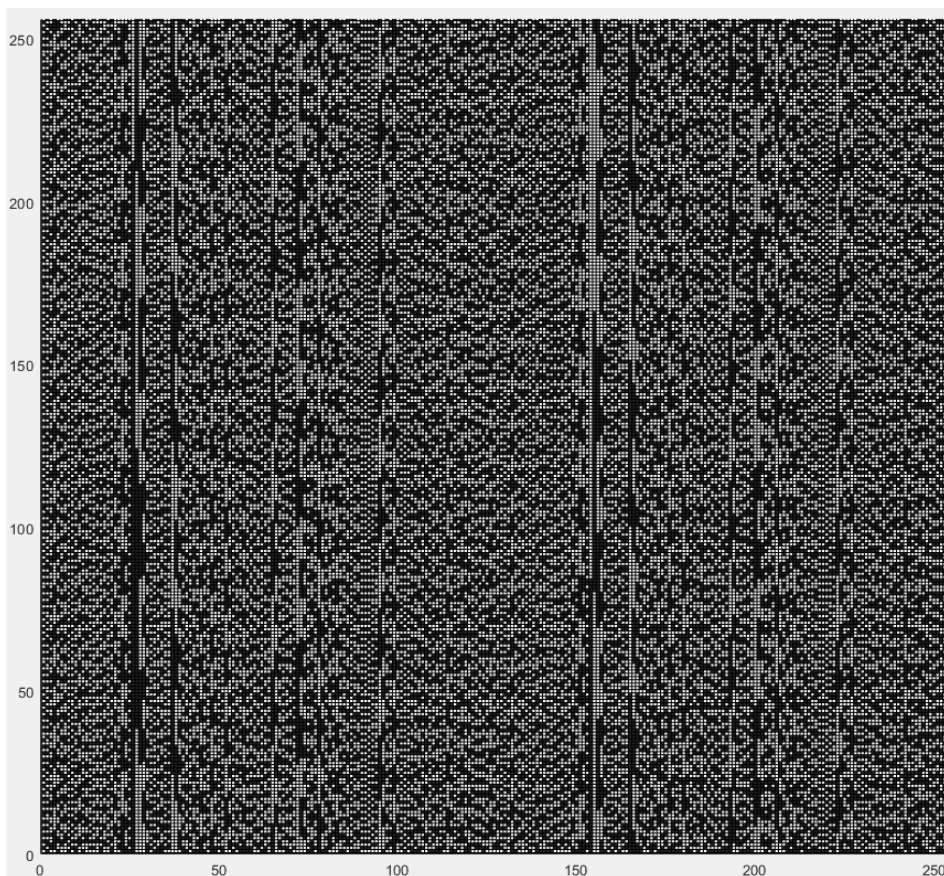
Step 0

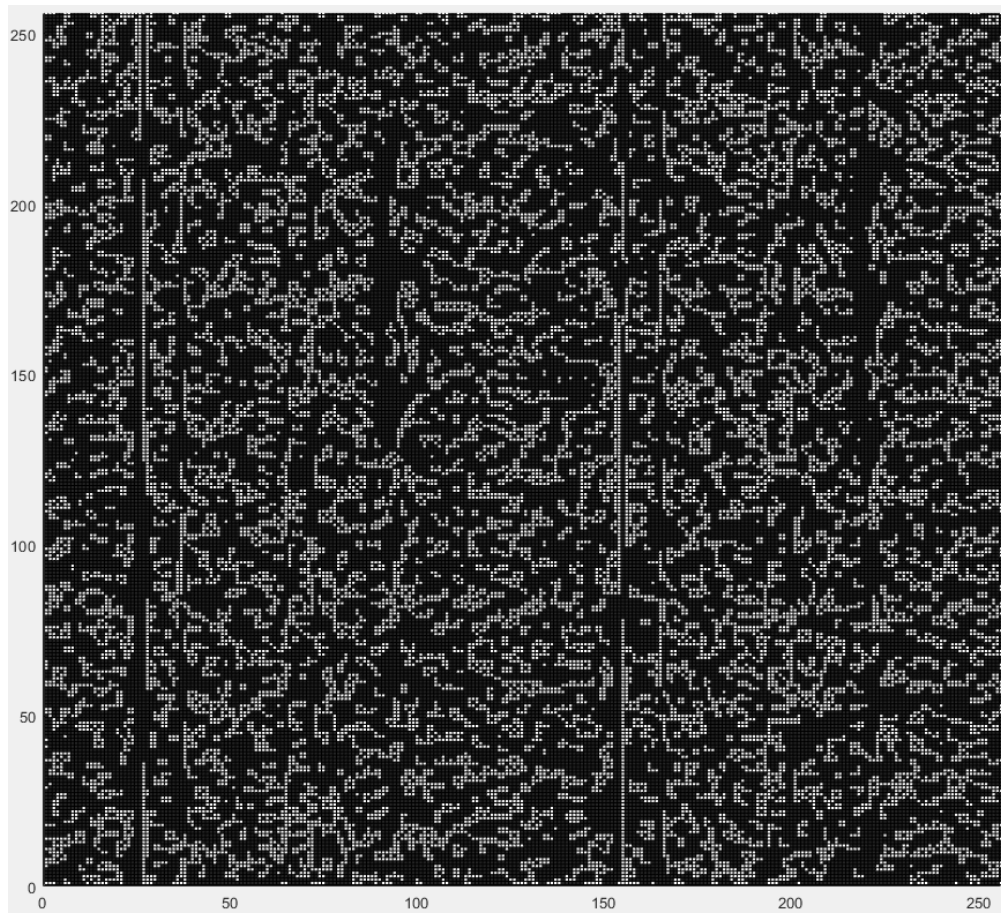


Step 1

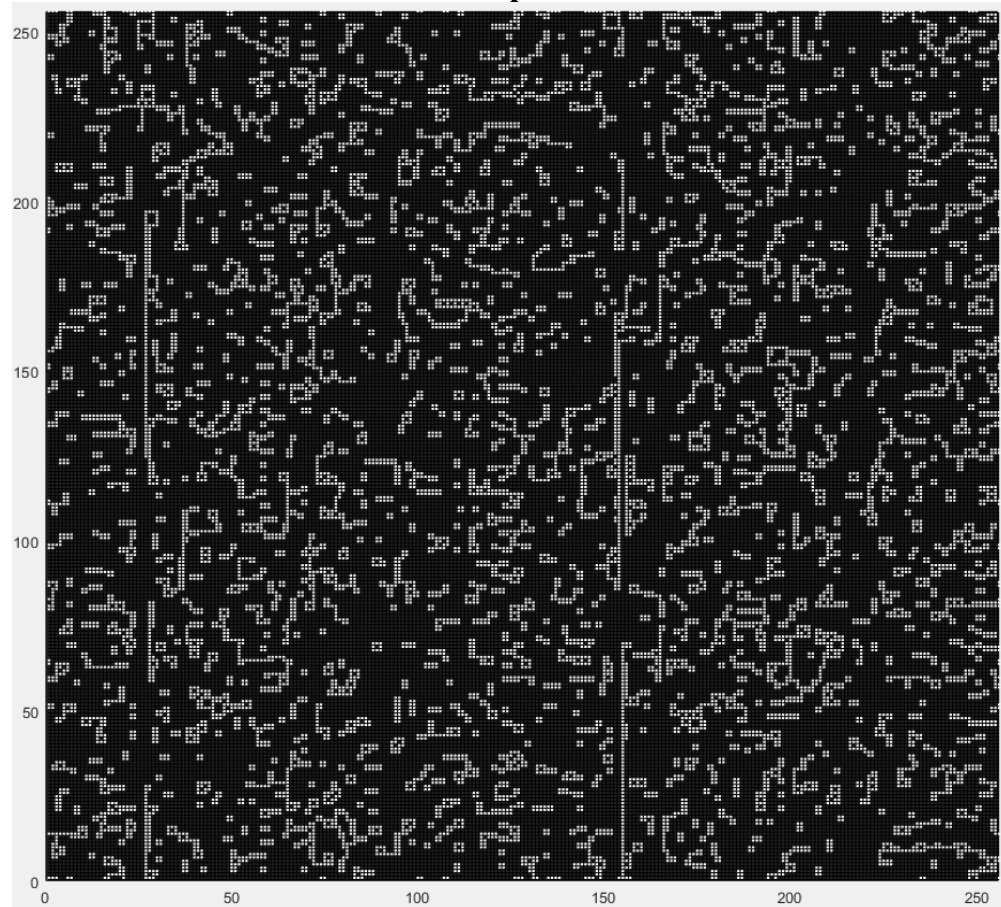
**Step 100**

256 x 256 Grid

**Step 0**

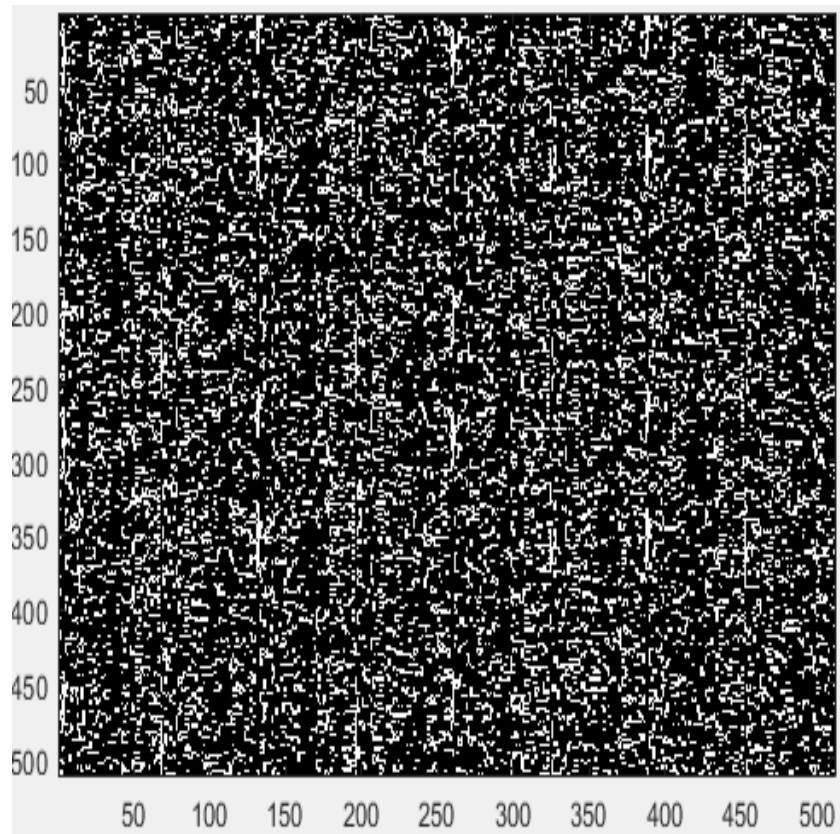


Step 1

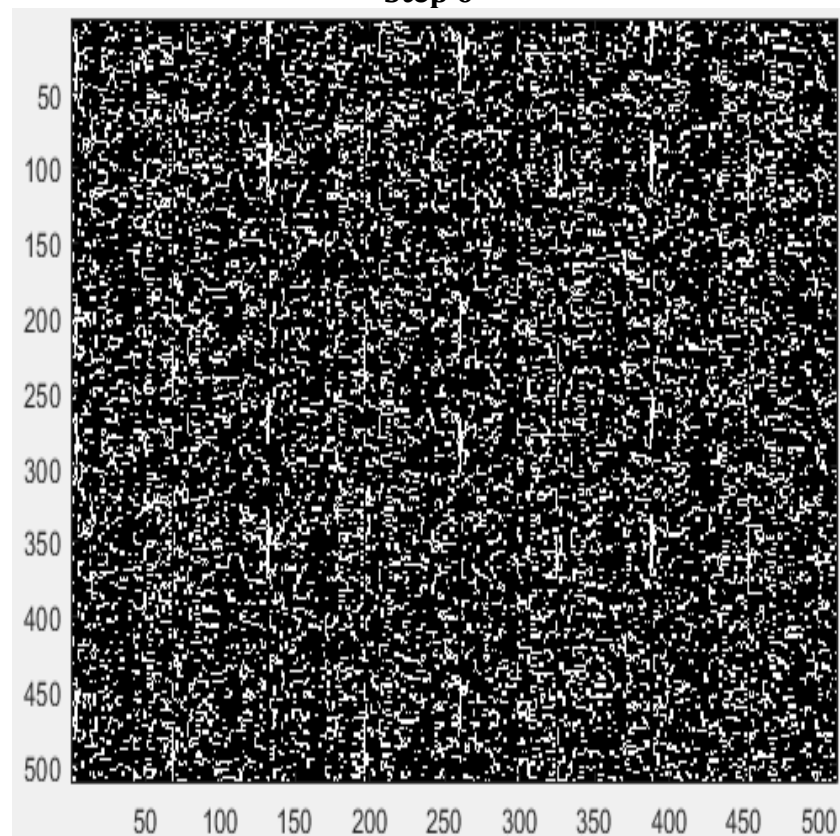


Step 100

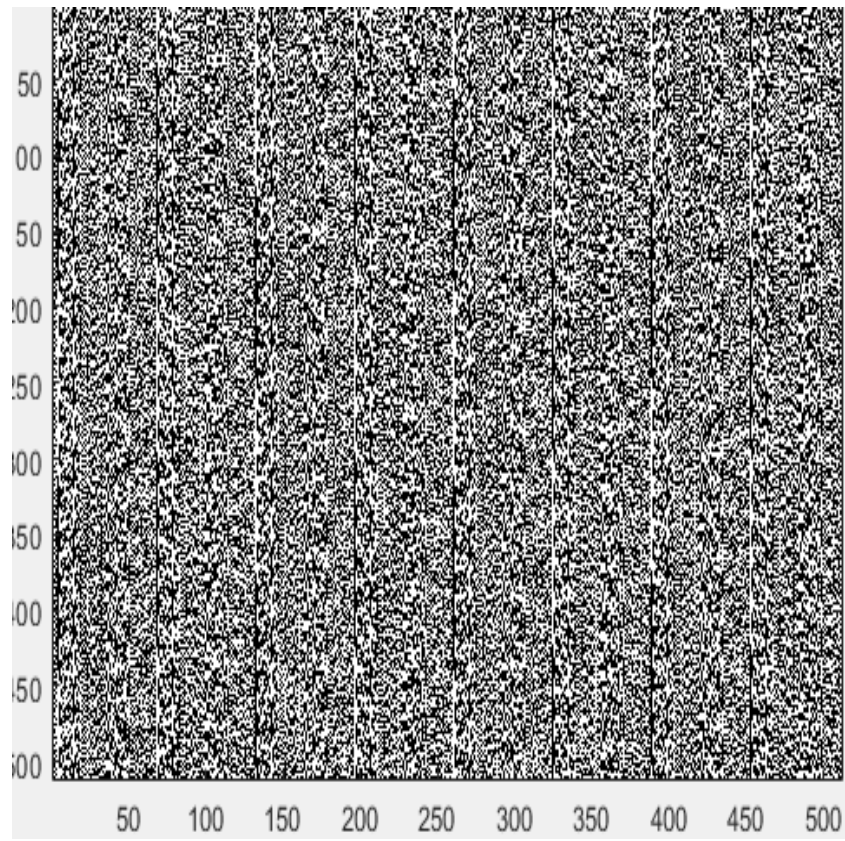
512 x 512 Grid



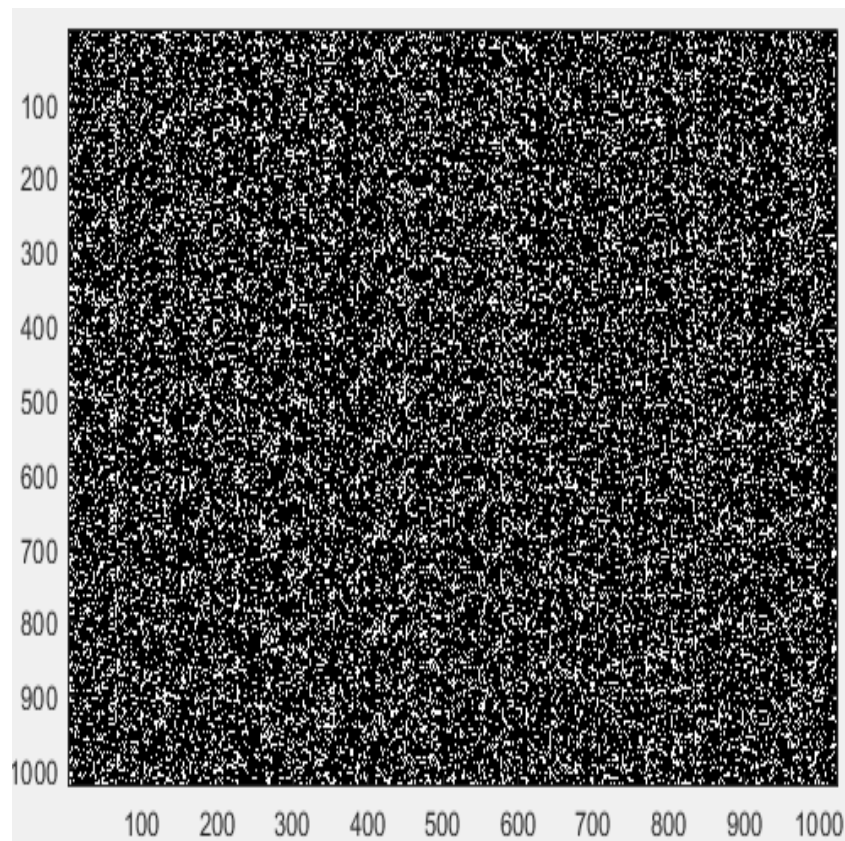
Step 0

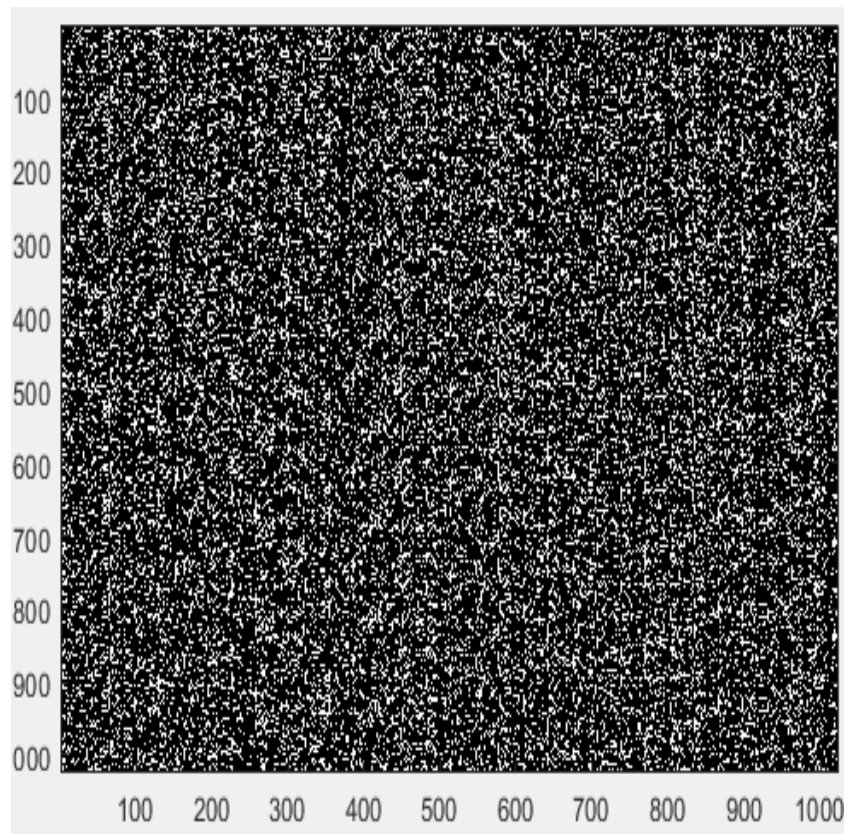
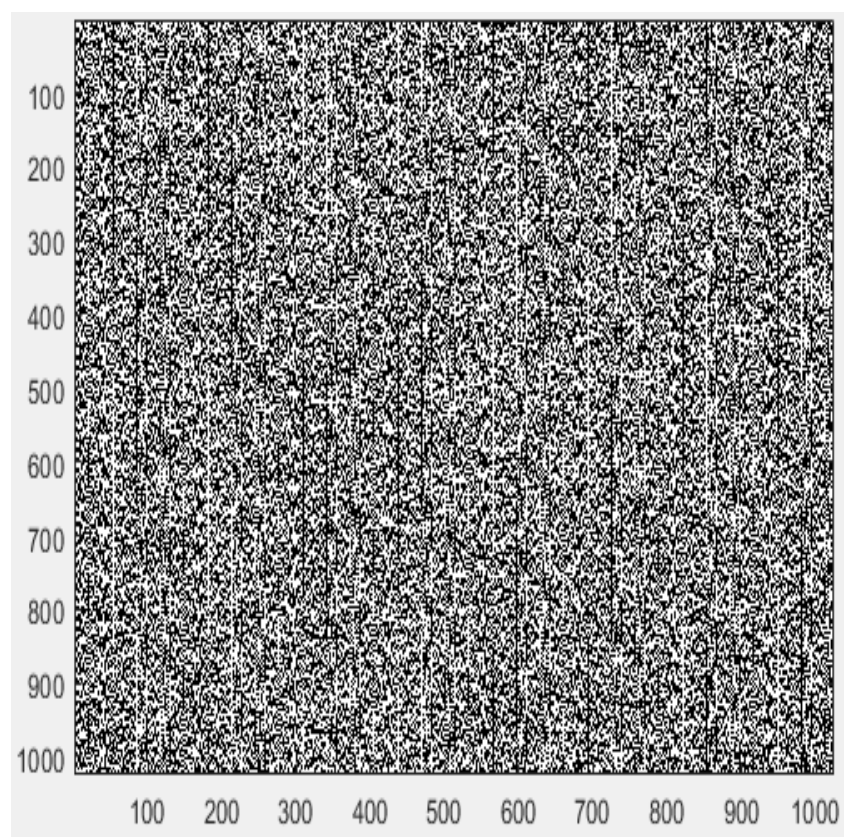


Step 1

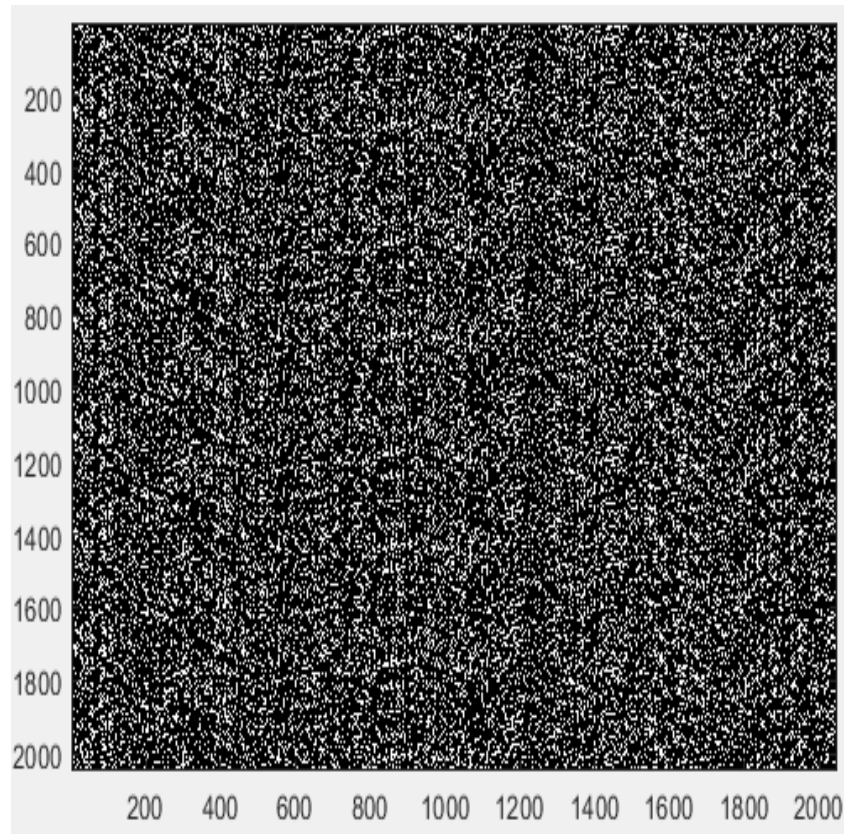
**Step 100**

1024 x 1024 Grid

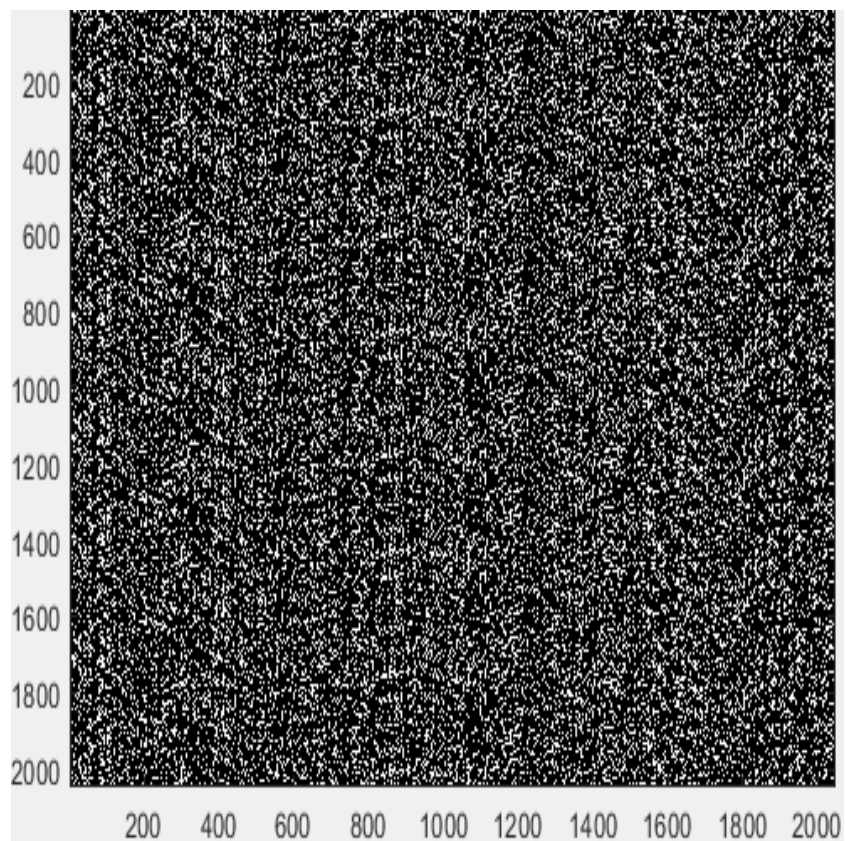
**Step 0**

**Step 1****Step 100**

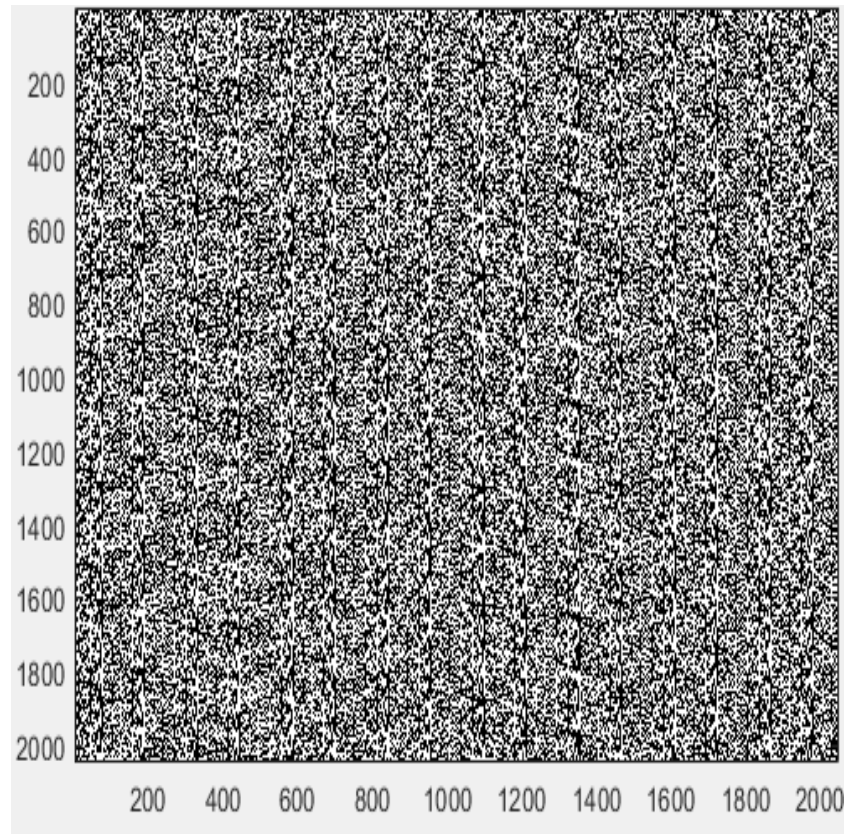
2048 x 2048 Grid



Step 0



Step 1



Step 100

----- END OF REPORT -----