Cellular Automata: Simulations Using Matlab

Stavros Athanassopoulos[1,2], Christos Kaklamanis[1,2], Gerasimos Kalfoutzos[1], Evi Papaioannou[1,2]
[1]Dept. of Computer Engineering and Informatics, University of Patras
[2]Computer Technology Institute and Press "Diophantus"
Patras University Campus, Building B, GR26504, Rion, Greece
e-mail: {athanaso, kakl, kalfount, papaioan}@ceid.upatras.gr

***Abstract*—This paper presents a series of implementations of cellular automata rules using the Matlab programming environment. A cellular automaton is a decentralized computing model providing an excellent platform for performing complex computations with the help of only local information. Matlab is a numerical interactive computing environment and a high-level language with users coming from various backgrounds of engineering, science, and economics that enables performing computationally intensive tasks faster than with traditional programming languages (such as C, C++, and Fortran). Our objective has been to investigate and exploit the potential of Matlab, which is simple mathematical programming environment that does not require specific programming skills, regarding the understanding and the efficient simulation of complex patterns, arising in nature and across several scientific fields, captured by simple cellular automata structures. We have implemented several cellular automata rules from the recent literature; herein we present indicative cases of practical interest: the forest fire probabilistic rule, the sand pile rule, the ant rule, the traffic jam rule as well as the well-known "Game of Life". Our work indicates that Matlab is indeed an appropriate environment for developing simulations for cellular automata models.**

Keywords-cellular automata; simulation; Matlab.

I. CELLULAR AUTOMATA

A cellular automaton (CA) is an idealization of a physical system in which space and time are discrete and the physical quantities take only a finite set of values. Informally, a cellular automaton is a lattice of cells, each of which may be in a predetermined number of discrete states (a formal definition can be found in [7]). A neighborhood relation is defined over this lattice, indicating for each cell which cells are considered to be its neighbors during state updates. Time is also discrete; in each time step, every cell updates its state using a transition rule that takes as input the states of all cells in its neighborhood (which usually includes the cell itself). All cells in the cellular automaton are synchronously updated. At time t = 0 the initial state of the cellular automaton must be defined; then repeated synchronous application of the transition function to all cells in the lattice will lead to the deterministic evolution of the cellular automaton over time. Many variations of this basic model exist: CA can be of arbitrary dimension, although one-dimensional and two-dimensional CA have received special attention in the literature. CA can be infinite or finite. Finite CA can have periodic boundaries (e.g., the opposite ends of a one-dimensional finite CA are joined together so the whole forms a ring). Updates can be synchronous or asynchronous. Transition rules can be deterministic or stochastic. Many other variations exist; those mentioned above are some of the most typical ones.

The concept of cellular automata was initiated in the early 1950's by John Von Neumann and Stan Ulam [18]. Von Neumann was interested in their use for modelling self-reproduction and showed that a CA can be universal. He devised a CA, each cell of which has a state space of 29 states, and showed that it can execute any computable operation. However, Von Neumann rules, due to their complexity, were never implemented on a computer. Von Neumann's research raised a dichotomy in CA research. On one hand, it was proven that a decentralized machine can be designed to simulate any arbitrary function. On the other hand, this machine (CA) can become as complex as the function it is intended to simulate.

Cellular automata have received extensive academic study into their fundamental characteristics and capabilities and have been applied successfully to the modelling of natural phenomena and complex systems [1], [3], [4], [13], [17], [24], [23]. Based on the theoretical concept of universality, researchers have tried to develop simpler and more practical architectures of CA that can be used to model widely divergent application areas. In the 1970, the mathematician John Conway proposed the (now famous) Game of Life [10], which received widespread interest among researchers. Since the beginning of the 80's, Stephen Wolfram has studied in much detail a family of simple one-dimensional cellular automata rules (known as Wolfram rules [24]) and has showed that even these simplest rules are capable of emulating complex behavior. Other applications include, but are not limited to, theoretical biology [2], game theory [19], and non-equilibrium thermodynamics [15].

The rest of the paper is structured as follows: Section II includes a brief description of Matlab as well as main reasons that motivated us for using it in our simulations. Simulations are presented in Section III. Section IV includes conclusion and plans for future work on cellular automata simulations using Matlab.

 ISBN: 978-1-61208-176-2

II. MATLAB

MATLAB is a numerical computing environment and fourth-generation programming language which allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. Although it was intended primarily for numerical computing, it also allows symbolic computing, graphical multi-domain simulation and model-based design for dynamic and embedded systems. It has been widely used in academia and industry by users coming from various backgrounds of engineering, science and economics. MATLAB was first adopted by researchers and practitioners in control engineering, and quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis, and is very popular amongst scientists involved in image processing [16].

Why we used Matlab for our simulations? Existing implementations of cellular automata have been developed using Java and C/C++. This selection has been supported by the graphical interface these programming languages offer as well as by their strict object-oriented programming nature. In this way, implementation of cellular automata can be a very efficient and effective development task. For our study, Matlab offers simplicity coupled with power; this mainly motivated us to use it for the implementation/simulation of cellular automata, i.e., of simple structures that can, however, model complex behavior and real-world patterns. Matlab neither requires nor focuses on particular programming skills; on the contrary, it provides an efficient tool for the researcher/user to simulate simple models without focusing on programming and easily conceive such complex patterns in practice – not only through some mathematically defined function (however, using appropriate toolboxes, Matlab code can be converted – if needed – to C/C++ code).

More specifically, cellular automata can be implemented using matrices of one or several dimensions. Matlab makes a quite appropriate environment since it offers a wide range of operations and functions particularly working on matrices. Moreover, the status of network cells can be easily represented using function surf(), while necessary diagrams and graphical representations can be produced - almost directly - using function plot(). Using Matlab only a single file per cellular automaton (i.e., per algorithm) is needed; this provides high flexibility in the experimentation and simplicity in the code execution process. Furthermore, syntax is simpler (than in involved programming languages) thus directly reflecting the simplicity of the rules according to which automaton cell status is altered. Such technicalities could be of high importance when it comes to communities of researchers not familiar with programming languages: they could easily deploy their model and see its behavior without having to spend extra resources for becoming programming experts. Of course, Matlab is a rather slow environment and Matlab programs require more computational power compared to Java or C++; this could be a drawback if our algorithms were to be used as parts of intense resource-requiring applications.

III. OUR SIMULATIONS

As already stated, the question that motivated our work is the following: Matlab is a "simple" programming environment that does not require a researcher/student to be a programming-expert to use it. Cellular automata can capture, via a small set of simple rules, very complex phenomena from the real world. Is Matlab efficient for simulations involving cellular automata?

We have implemented several CA rules from the recent literature: the Wolfram's 184 rule, rules for probabilistic cellular automata, the Q2R rule, the annealing rule, the HPP rule, the sand pile rule, the ant rule, the traffic jam rule, the solid body motion rule, the "*Game of Life*". Detailed description of these rules can be found in [7].

Herein, we present in detail five indicative cases of practical interest we simulated (and used for teaching purposes in the Theory of Computation lab of our department): Probabilistic Cellular Automata rules for forest fire models, the Sand Pile rule, the Ant rule, the Traffic Jam rule and the John Conway's Game of Life.

Matlab Version 7.0.0.19920 (R14) has been used for implementation. Simulations have been executed on a system using an Intel Core i3 530 processor (2.93GHz, 6144MB DDR3 RAM), running Windows 7 Premium 32-bit operating system. For the graphical representation of the behavior of simulated models, function surf() has been used (full size figures can be found at [25]).

A. Implementation of a probabilistic rule for Burning Forest

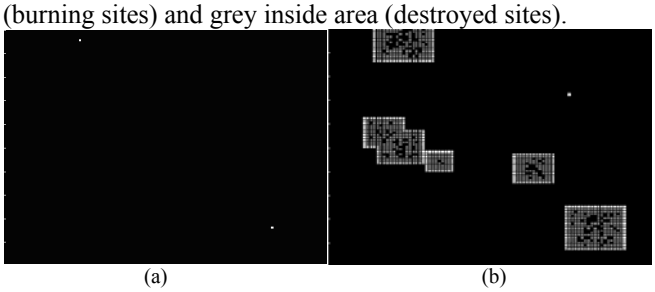
Probabilistic Cellular Automata (PCA) are ordinary cellular automata where different rules can be applied at each cell according to some probability [24]. An interesting and simple example of a PCA model is a probabilistic rule for Burning Forest. The cellular automaton used for simulation uses a (nxn) grid, representing the forest, and a Moore neighborhood. Cells correspond to trees and can be in one of the following three states: green tree (1), empty site (2), burning tree (3). Initially, all cells are in state (1) (i.e., contain a green tree). Cell states are updated according to the following rules presented in detail in [5], [8]:

- A burning tree becomes an empty site.
- A green tree becomes a burning tree if at least one of its nearest neighbors is burning.
- At an empty site, a tree grows with probability p.
- A tree without a burning nearest neighbor becomes a burning tree in one time step with probability f.

 ISBN: 978-1-61208-176-2

At each time step, every cell is assigned a new random value (in [0,1]) for fire (f) and birth (p) probability. A green tree becomes a burning tree when f is greater than a threshold value set to 0.001. A new tree grows in an empty site when p is greater than a threshold value set to 0.1. These threshold values for f and p, once set remain the same throughout a single execution. Threshold value for f has been chosen to be sufficiently small so that in a large grid only few fires can start. Threshold value for p has been chosen to be greater than this for f so that new trees can grow and simulation can continue.

The following figures show instances of the simulation using a grid of size 200x200. In the beginning (Fig. 1a) two fires (white areas) have started in the forest (black area). Fire starts spreading among green trees, leaving empty sites behind (grey areas) (Fig. 1b). The fire spreading pattern looks like growing circular discs with a white outline (burning sites) and grey inside area (destroyed sites).



(a) (b)

Fig. 1: Two fires have started in the forest (white sites) (a). The fire is spreading among green trees, turning them to empty sites (b).

B. Implementation of the Sand Pile rule

The physics of granular materials has recently attracted CA-related research interest. It is possible to devise a simple cellular automaton rule to model basic piling and toppling of particles like sand grains [7]. The idea is that grains can stack on top of each other if this arrangement is stable. Of course, real grains do not stand on a regular lattice and the stability criteria are expected to depend on the shape of each grain. Despite the microscopic complexity, the result is sand piles that are too high to topple.

Toppling mechanisms can be captured by the following cellular automaton rule: a grain is stable if there is a grain underneath and two other grains preventing it falling to the left or right (Fig. 2). Assuming a Moore neighborhood, the rule implies that a central grain will be at rest if the south-west, south and south-east neighbors are occupied. Otherwise, the grain topples downwards to the nearest empty cell.

Figure 2: The top grain will not move.

The cellular automaton used for simulation uses a (nxn) grid and a Margolus neighborhood which gives a simple way to deal with the synchronous motion of all particles [20]. Informally, when Margolus neighborhood is used, the lattice is divided in disjoint blocks of size 2x2; each block moves down and to the right with the next generation, and then moves back [21]. Cells can be in one of the following two states: grain of sand (1), empty cell (0). Initially, sand grains are placed randomly on the grid (no additional grains appear during the evolution of the cellular automaton). Cell states are updated according to the following rule [7], which is also presented graphically in Fig. 3:

Current state	1000	0100	1010	1001	0110	0101	1110	1101	1100 (p)	1100 (1-p)
Next state	0010	0001	0011	0011	0011	0011	1011	0111	0011	11100

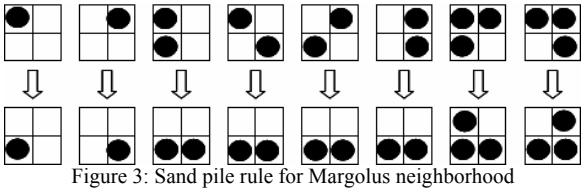


Figure 3: Sand pile rule for Margolus neighborhood

The configuration in which the upper part of a block is occupied by two particles while the lower part is empty, is not listed in the above image, although it certainly yields some toppling. When this configuration occurs, we adopted the probabilistic evolution rule shown in Fig. 4 in order to produce a more realistic behavior: some friction may be present between grains and some arches may appear to delay collapse. Of course, the toppling of other configurations could also be controlled by a random choice.

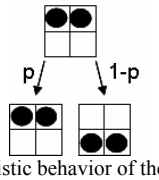
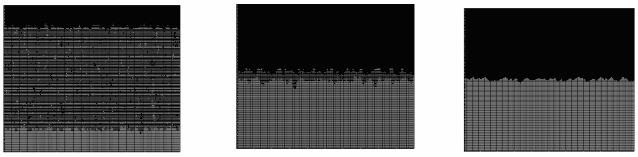


Figure 4: Probabilistic behavior of the sand pile rule [7]

In this simulation, p has been set to 0.5, i.e., two neighboring grains can equiprobably either fall (filling the cells bellow them) or remain at rest.

Fig. 5a, 5b and 5c show simulation instances. Initially, all grains are falling, except those at the bottom which remain at rest. The Margolus neighborhood does not affect grains at the grid boundaries, so they also remain at rest. The sand pile is growing and the number of falling grains decreases (Fig. 5b). The simulation terminates when there are no more grains to fall (Fig. 5c).



(a) (b) (c)

Figure 5: The initial state of the lattice (a). The growing sand pile due to falling grains (b). Finally, a sand pile is created (c)

C. Implementation of the Ant rule

Langton's Ant [13, 14] follows extremely simple rules and initially appears to behave chaotically, however after a

certain number of steps a recurring pattern emerges. Langton's Ant models the true behavior of ants in nature: a moving ant tends to leave pheromones behind it. All other ants moving in the same area can sense that substance and follow the motion of the first ant.

The rule simulates the following idea: an ant sits on a cell of a grid where all other cells are initially empty. It moves into a neighboring cell and does one of two things, based on the color of the cell:

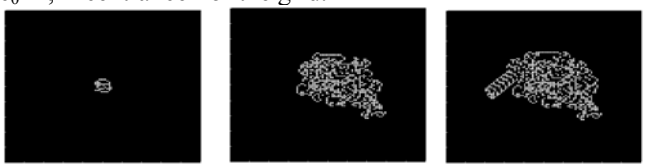
– If the square is white, it turns 90 degrees to the left and colors the square grey
– If the square is grey, it turns 90 degrees to the right and colors the square white

The movement is continued in the same fashion, ad infinitum. The interesting thing about this is that after a fixed number of steps, the ant builds a highway and hotfoots it into infinity. The motion of the ant in this highway is not linear; it rather looks like the pattern of operation of a sewing machine. Although the ant rule seems to be very simple, it drives the ant to a chaotic state. This feature also shows the power of modeling systems with cellular automata: even though the cellular automata rules are very simple, they can implement very complex behaviors.

The cellular automaton used for simulation uses a (nxn) grid and a Von Neumann neighborhood; a von Neumann neighborhood is composed of the four cells orthogonally surrounding a central cell on a two-dimensional square lattice [12]. A cyclic neighbourhood has been used for cells at the lattice boundaries: when an ant reaches the lattice boundaries, it returns to the lattice simulating the existence of a second ant. Cells can be in one of two states: ant (1), empty cell (0). Initially, all cells are empty (state 0) apart from one cell (state 1) which contains the ant. Cell states are updated according to the following rules:

– $n_i (r + c_i, t + 1) = \mu\, n_{i-1} (r, t) + (1 - \mu)\, n_{i+1} (r, t)$
– $\mu (r, t + 1) = \mu (r, t) \oplus n_1(r, t) \oplus n_2(r, t) \oplus n_3(r, t) \oplus n_4(r, t)$

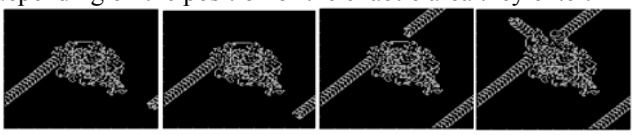
where n_i: new state, r: current cell, c_i: current direction, t: current time step, μ: cell color (1=white, 0=black). Initially, c_0=4, r=central cell of the grid.



(a) (b) (c)

Figure 6: (a) The ant starts its journey from the centre of the lattice. (b) Chaotic situation due to the ant movement. (c) Ant's highway.

In our simulation, an ant starts its journey from the central cell of a 100x100 grid (Fig. 6a). All cells are initially black; the ant turns them white as it moves over them. After approximately 7000 steps, the ant is trapped in a chaotic situation (Fig. 6b). After approximately 10000 steps, the ant creates its way out of the chaotic situation, building its highway and moving away from its initial position (Fig. 6c). As soon as the ant reaches the lattice boundary it returns to the lattice from a different position as a "second" ant, which has just entered the area. This second ant continues moving on the highway, just like the first one (Fig. 7a), moves towards the chaotic area (created by the first ant) (Fig. 7b) and starts moving irregularly (Fig. 7c). The "second" ant "senses the pheromones" of the first ant and escapes the chaotic situation faster than the previous ant. Ants can either create their own highways (Fig. 7d) or follow existing ones depending on the position of the chaotic area they enter.



(a) (b) (c) (d)

Figure 7: the movement of a second ant

D. Implementation of the Traffic Jam rule

Cellular automata models for road traffic have received a great deal of interest. One-dimensional models for single-lane car motions are quite simple and elegant [6]. The road is represented as a line of cells: each cell is either occupied by a vehicle or not. All cars travel in the same direction. Their positions are updated synchronously, in successive iterations (discrete time steps). During the motion, each car can be at rest or jump to the nearest-neighbor site, along the direction of motion. The rule is that a car moves only if its destination cell is empty. This means that the drivers are short-sighted and do not know whether the car in front will move or whether it is also blocked by another car. Therefore the state of each cell s_i is entirely determined by the occupancy of the cell itself and its two nearest neighbors s_{i-1} and s_{i+1}. The motion rule can be summarized in the following table, where all eight possible configurations $(s_{i-1}\ s_i\ s_{i+1})_t \rightarrow (s_i)_{t+1}$ are given [6]:

111	110	101	100	011	010	001	000
1	0	1	1	1	0	0	0

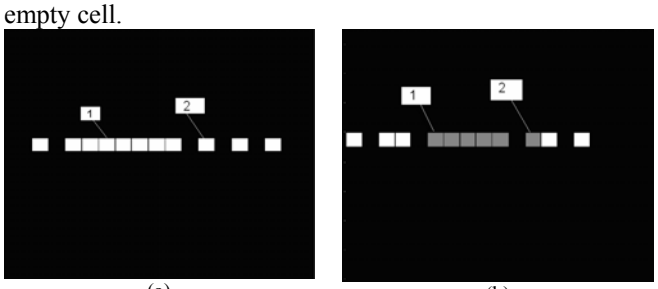
This simple dynamics captures an interesting feature of real car motion: traffic congestion. This cellular automaton rule turns out to be Wolfram's rule 184 [6].

The cellular automaton used for simulation uses a line and a one-dimensional neighborhood. Cells can be in one of three states: empty cell (0), stopped car (1), moving car (2). Initially, cars are placed randomly in line cells. Cell states are updated according to the following rule:

– $n_i(t+1)=n_i^{in}(t)(1-n_i(t))+ n_i(t)\, n_i^{out}(t)$,

where $n_i(t)$ denotes the car occupation number (n_i=0: free site, n_i=1: a car is present at site i). $n_i^{in}(t)$ denotes the state of the source cell, i.e., that from which a car may move to cell i. Similarly, $n_i^{out}(t)$ indicates the state of the destination cell, i.e., that the car at site i would like to move to. The rule implies that the next state of cell i is 1 if a car is currently present and the next cell is occupied, or if no car is currently present and a car is arriving.

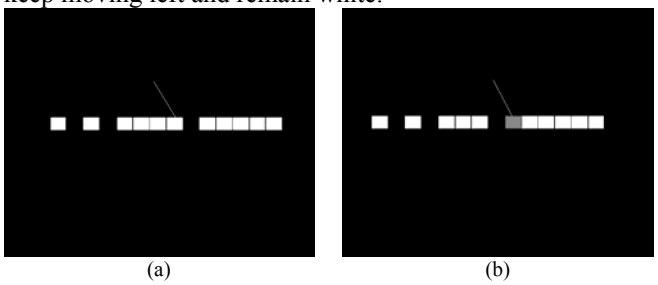
A car is moving or not according to its "speed", a variable taking random values in [0,1] that change in each time step. If a car has a "speed" lower than a threshold value set to 0.05, then it stops for one time step. When a car reaches the leftmost cell of its row, it is injected in the rightmost cell of the lattice in the same row and keeps moving in loops. Fig. 8a shows a normal traffic instance where all the cars are moving from right to left by one cell per step. White cars are moving; grey cars have stopped. In Fig. 8b, cars 1 and 2 stop. When car 1 stops, all following cars also stop (since there are no empty cells between them) and turn grey. Cars in front of car 1 keep moving left because no preceding car has stopped. When car 2 stops, there is an empty cell behind it. This is why the following cars remain white and keep moving left, covering every empty cell.



(a) (b)

Figure 8: Normal traffic (a) and traffic with cars that are not moving (b).

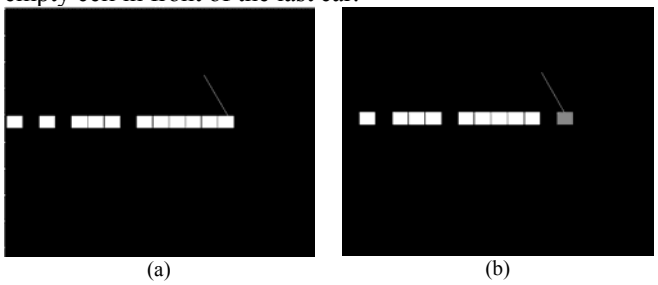
In Fig. 9a, another instance of normal traffic is shown. The car pointed by the arrow stops and becomes grey (Fig. 9b). There is an empty cell behind it, so all cars that follow keep moving left and remain white.



(a) (b)

Figure 9: Normal traffic (a) and then a car stops (b)

Finally, the last car (Fig. 10a) stops (and becomes grey in Fig. 10b). All other cars keep moving left leaving an empty cell in front of the last car.



(a) (b)

Figures 10: Normal traffic (a) and then a car stops (b)

E. John Conway's Game of Life

The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway in 1970 [10]. The game is a zero-player game, meaning that its evolution is determined by its initial state (called pattern), requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves. 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 (white-1) or dead (black-0). Every cell interacts with its eight neighbours (Moore neighbourhood), which are the cells that are horizontally, vertically, or diagonally adjacent. Cell states are updated according to the following rule:

- Any live cell with fewer than two live neighbours dies, as if caused by under population.
- 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 overcrowding.
- Any cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial pattern placed in the middle cells of the grid 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 by deaths occurring simultaneously, and the discrete moment at which this happens is sometimes called a tick; in other words, each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

Fig. 11 shows simulation snapshots for 6 different initial patterns: cell row (Fig. 11a), glinder (Fig. 11b), small explorer (Fig. 11c), explorer (Fig. 11d), lightweight spaceship (Fig. 11e), tumbler (Fig. 11f).

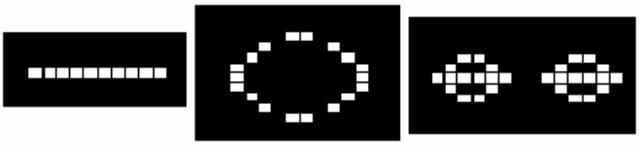


Figure 11a: Cell Row

Figure 11b: Glinder

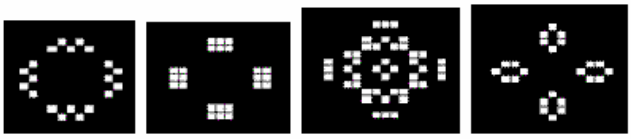


Figure 11c: Small explorer

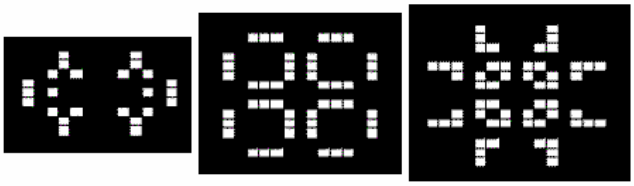


Figure 11d: Explorer

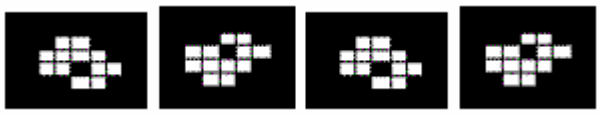


Figure 11e: Lightweight spaceship

Figure 11f: Tumbler

IV. CONCLUSION AND FUTURE WORK

We have simulated several popular cellular automata rules of practical interest using Matlab. Our simulations yield evolution patterns in accordance with those expected from corresponding rules and similar to those obtained so far using Java or C/C++.

Our work indicates that Matlab is indeed an appropriate environment for developing compact code for simulations involving cellular automata, even though it does not always guarantee high simulation speeds. It does not require specific programming skills and therefore it offers the flexibility to non-programming-expert researchers and/or students to experiment and understand in practice complex patterns captured by simple cellular automata structures.

Our current ongoing work investigates the potential of Matlab for simulations involving cellular automata for problems related to energy-efficient communication in Wireless Sensor Networks.

ACKNOWLEDGMENT

This work has been partially supported by EU under the ICT-2010-258307 project EULER and by EU and the Greek Ministry of Education, Lifelong Learning and Religious Affairs under the project “Digital School” (296441).

REFERENCES

[1] S. Bandini. Guest Editorial - Cellular Automata. Future Generation Computer Systems, 18:v-vi, August 2002.

[2] M. Boerlijst and P. Hogeweg. Spiral wave structure in pre-biotic evolution: hypercycles stable against parasites. Physica D, 48(1):17–28, 1991.

[3] A. W. Burks. Essays on Cellular Automata. Technical Report, Univ. of Illinois, Urbana, 1970.

[4] P. Pal Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chatterjee. Additive Cellular Automata - Theory and Applications, volume 1. IEEE Computer Society Press, CA, USA, ISBN 0-8186-7717-1, 1997.

[5] P. Bak, K. Chen, C. Tang. A forest-fire model and some thoughts on turbulence. Physics Letters A, Vol. 147, Issues 5-6, pp. 297-300, 1990.

[6] B. Chopard, P. O. Luthi, and P-A. Queloz. Cellular Automata Model of Car Traffic in a Two-Dimensional Street Network. Journal of Physics A: Mathematical and General, 29, pp. 2325–2336, 1996.

[7] B. Chopard and M. Droz. Cellular Automata Modeling of Physical Systems, Cambridge University Press, 1998. ISBN 0-521-46168-5.

[8] B. Drossel, F. Schwabl. Self-organized critical forest-fire model. Physical Review Letters, Vol. 69, Issue 11, pp. 1629–1632. 1992.

[9] N. Ganguly, B. K. Sikdar, A. Deutsch, G. Canright, and P. Pal Chaudhuri. A survey on cellular automata. Technical report, Centre for High Performance Computing, Dresden University of Technology, December 2003.

[10] M. Gardner Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life". Scientific American, 223. pp. 120-123, 1970. ISBN 0894540017.

[11] R. Goering. Matlab edges closer to electronic design automation world. EE Times, 10/04/2004 (http://www.eetimes.com/electronics-news/4050334/Matlab-edges-closer-to-electronic-design-automation-world).

[12] L. Gray. A Mathematician Looks at Wolfram's New Kind of Science. Not. Amer. Math. Soc. 50, 200-211, 2003.

[13] C. G. Langton. Self-reproduction in cellular automata. Physica D: Nonlinear Phenomena, Volume 10, Issues 1-2, pp. 135-144, 1984. ISSN 0167-2789, 10.1016/0167-2789(84)90256-2.

[14] C. G. Langton. Studying artificial life with cellular automata. Physica D: Nonlinear Phenomena 22 (1-3): 120–149, 1986.

[15] M. Markus B. Hess. Isotropic cellular automaton for modelling excitable media. Nature, 347(6288):56–58, 1990.

[16] C. Moler. The Origins of MATLAB. December 2004. Retrieved April 15, 2007.

[17] M. Mitchell, P. T. Hraber, and J. P. Crutcheld. Revisiting the Egde of Chaos: Evolving Cellular Automata to Perform Computations. Complex Systems, 7, pp. 89-130, 1993.

[18] J. V. Neumann. The Theory of Self-Reproducing Automata. A. W. Burks (ed), Univ. of Illinois Press, Urbana and London, 1966.

[19] M. Nowak and R. May. Evolutionary games and spatial chaos. Nature, 359(6398):826–829, 1992.

[20] J. Schiff. 4.2.1 Partitioning Cellular Automata. Cellular Automata: A Discrete View of the World, Wiley, pp. 115–116, 2008.

[21] T. Toffoli, N. Margolus. II.12 The Margolus neighborhood. Cellular Automata Machines: A New Environment for Modeling, MIT Press, pp. 119–138, 1987.

[22] S. Wolfram. A New Kind of Science. Champaign, IL: Wolfram Media, pp. 29-30, 52, 59, 317, and p. 871, 2002.

[23] S. Wolfram. Cellular Automata and Complexity. World Scientific, Singapore, 1994. ISBN 9971-50-124-4 pbk.

[24] S. Wolfram. Theory and Applications of Cellular Automata. World Scientific, Singapore, 1986. ISBN 9971-50-124-4 pbk.

[25] http://www.ceid.upatras.gr/papaioan/CA/figs/index.html, November 15, 2011.

 ISBN: 978-1-61208-176-2