Polanyiana 2017 (26) 1-2.

Emergent Computing and the Embodied Nature of Computation

Mihály Héder

Department of Philosophy and History of Science

Budapest University of Technology and Economics

e-mail: mihaly.heder@filozofia.bme.hu

Abstract: If we define computing as the typical activity of computers – in contrast to other approaches that define computation as an abstract mathematical process – it will become evident that in some cases the result of computation is not only a function of the algorithm executed but also of certain physical processes. For example, random numbers, which are used in many algorithms, are usually generated by sampling the physical environment of the computer. This may sound trivial, but as we will see some philosophical arguments around emergence seem to rely on some sort of platonic conception of computation and thereby draw conclusions that do not apply to real-world computation. In other words, we should not forget that computing is embodied, and different embodiments lead to different computation results. This article investigates what can be said about the question of emergent computing in light of computation's embodied nature.

Key words: computers, emergent computing, epistemic emergence, *Game of Life*, ontological emergence.

1. Introduction

The aim of this article is to explore what does *emergent computing* mean, and in what circumstances can computing be said to be *emergent*. Emergence is a centuries-old philosophical problem. In addition, in the last one or two decades, engineers, while trying to solve hard computational problems, invented a solution which they called *emergent computing*. This coinage was not based on a dialog between philosophers and engineers. It was the result of the internal evolution of computer science independent of the philosophical

debates. Thus, there were, at least, two main approaches to be considered in this inquiry about emergence and computing. But this does not necessarily mean that *emergent computing* and the philosophical notion of *emergence* are incompatible with each other.

There is some literature discussing how emergent computing fits into the original philosophical debate. Most of these accounts drew the conclusion that computing can be *epistemically* emergent (e.g. Cariani 1991), but not *ontologically*. For most of the authors, the epistemic nature of emergent computing meant that, for an observer, the computation produces a surprising, complex phenomenon. But this phenomenon was identical with a set of more primitive building blocks, which were manipulated by simple operations. So, for these authors, the complex phenomenon was not a novel entity existing on its own, no matter what the impression of the observer was.

This view was best summarized by Barry S. Cooper (2011). He described emergent computing in the following way: 1) An engineer creates a computation, using a program on level L1. 2) An observer recognizes a complex phenomenon, or behaviour, like the typical complex dynamics of a stock market. Cooper called this an L2 level observation. 3) For the observer, it is not evident how the instructions of the L1 program lead to the phenomenon on L2. Moreover, sometimes the connection between L1 and L2 is not immediately apparent, even to the engineers themselves. What follows is that an L2 phenomenon is nothing more than an arrangement of elements on L1, hence a phenomenon on L2 does emerge in an *epistemic* sense.

But the discussion of emergent computing began much earlier. Cariani (1991) characterized emergent computing as a process in which global forms are created through micro-deterministic computations. This means that the emergent phenomenon is formed from bottom to top following certain

construction rules. Thus, the result is fully determined by the building blocks. Still, the result can be said to be emergent in the sense that the creator only had a model at the basic level. The complex phenomenon, that is, the result of the computation, is not part of the model and cannot be expressed by its elements and operators. In other words, this theory also described epistemic emergence.

A good case in point explaining Cariani's theory was an experiment published by Walshe et al. (2008). The authors created a simplified computational model that represented the inner workings of cells, cell-cell relations and cell-active substance relations in the human body. The model did not contain the majority of the known parameters of a cell – each cell was expressed by a vector, in which the first dimension described the cell's position in space, and some additional dimensions described how the cell interacted with active substances. In this way, a simple, multi-dimensional vector space was created in which the distance between two cells could be calculated based on their positions and chemical properties. With the addition of a set of rules that describes what should happen when cells are within a certain distance of each other, the model was ready to simulate cell population dynamics.

This model was used to simulate what happens in a human body when it is infected with the HIV virus. They ran the simulation with many different parameter sets, and found that the function of the number of Killer-T immune cells over time changes in very similar ways to those few real infections that have been documented since their very early stages. The shape of these functions is quite non-trivial, yet the numbers are very similar in the simulation and the real cases. This suggests that the researchers were able to create a simulation that behaves like a real cell population; therefore, it is possible to use the simulation to test different medications. According to Walshe et. Al., the behaviour of the simulated cells is emergent – one can best

understand what they mean by referring to Cariani's emergence-to-a-model theory.

According to other approaches, the term *emergent computing* describes the processes that occur in distributed and decentralized computing architecture, a long known, but only recently trending method of computer engineering. Relying on the work of Stephanie Forrest (Forrest 1990), Wim Hordijk (Hordijk 1999) defined emergent computing as global information processing that takes place in decentralized systems that extend in space. In a special issue of *ERCIM News*, Ruskin and Walshe gave another definition of *emergent computing*, calling it a complex process that is a "non-linear" combination of more simple processes (Ruskin and Walshe 2006).

There were additional computations that were claimed to be emergent in one way or another. For instance, in Conway's *Game of Life*, a simple binary matrix can exhibit exciting self-maintaining creatures, forms that shoot other forms, by using only three basic rules (Gardner 1970). By using evolutionary algorithms, a simple fitness function, and the basic rules of selection, mutation, and recombination, useful applications can be made. For example, according to a NASA plan described by Curtis et al. (2003), about 1000 micro-satellites could map an asteroid field in a concert of effort, yet each satellite would follow only a simple set of rules.

This quick review of the applications of emergent computing and their results also reveals a stereotypical view of computing itself. This view suggests that, the computer can be paused at a certain step, and its internal state can be accessed. Moreover, it is often said that the operation of a computer is nothing more than the execution of a chain of formalized commands, each one being determined by the previous one. In this model, emergence can, indeed, only occur in the sense that the appearance of a complex behaviour or form

surprises the observer. Such surprise can either come from a limited knowledge about the momentary working of the system or from a limited knowledge about how the mass of basic elements affect the operation of the system.

This article will argue that this view of computers and computation is inaccurate. It will suggest that, replacing our current view of computers with a more accurate one would enable a richer and more exciting interpretation of emergent computing. I will show that, a computer, at a given time, cannot be fully described by the contents of its memory and the architecture thereof. The physical state of the computer and its environment should also be taken into account when describing computers and computation, because these factors contribute to the working of computers. This does not mean that one could not reduce a complex phenomenon to the bits of a computer's internal state at a given time. (We can call this *synchronous reduction*, following Paksi's (2011) terminology). However, one could not reduce the bits of the current state to the bits of the previous state and the architecture (*diachronous reduction*). This will be important in the evaluation of evolutionary computations.

2. Embodied computation

Let us start with the axiom that every computation that has actually been carried out is embodied somehow, either in a living organism, or in a machine. The next question is, if one calculates the computation 2 + 2 in her head, and then, by pressing "2", "+", "2" on a calculator, are the two computations identical or not? If they are identical, what does exactly create the connection between them? In the philosophy of mathematics, it is well known that many mathematicians think that, there are abstract mathematical calculations that are governed by abstract principles. The actual calculations are implementing

the abstract 2 + 2 in one way or another. This is the Platonist understanding of mathematics in a very small nutshell (for more, see James Robert Brown 2008). Of course, the same abstract calculation can be realized multiple times, and the result is always the same if the realization is done right (of course, other properties of the computation, e.g. its speed, might vary). As a corollary, the mathematical analysis always describes the abstract calculation or computation, and not the physical realization.

But actual calculations are not only implementations of abstract mathematical calculations. They can be extremely useful in analyzing physical processes. If we can interpret a certain feature of a physical process as a number, and a related physical operation as a mathematical operator, then we can predict that, for instance, by mixing 1 dl of water with 4 dl of wine, we get 5 dl of liquid. That is, we can make predictions, analyses, and plans about the physical world using mathematics.

The connection between an abstract calculation and its physical realization is not always as close as in the case of simple addition. For example, a *stable Markov chain*[1] cannot predict the specific state of the process at a given time in the future, only the possibilities of certain states.

Following this train of thought, one quickly recognizes that the examples of emergent computing discussed above are very different from each other in the sense of mathematical definability. In case of the *Game of Life*, we should get the same results with every implementation as long as the implementation

[1]The *Markov chain* is a common tool in computing theory for describing certain stochastic processes that are characterizable by discrete states, and in which future states are only dependant on current states but not on past ones. The *Markov property* is the memoryless property of these stochastic processes.

follows the same rules. But the case of emergent computing (as we will see in the next part) is entirely different. The possible mathematical descriptions offer very limited descriptions which only address the results of computation. Still, these computer applications are dedicated to the creation of artifacts with very specific properties.

The problem of embodied computing – like the problem of emergence – was explored by computer scientists independently from the philosophical discussion of embodiment. One reason that scientists' attention turned to embodied computing was that certain problems have risen that endangered the fast pace of the evolution of computing hardware in recent decades. As MacLennan (2008) described the situation, while earlier every bit was represented by a very large number of physical particles, this ratio has started to decrease in the latest systems. Systems in which the number of bits and the number of particles that store the bits are increasingly close have become a real prospect in the near future.

For researchers working on such a small scale and tackling the properties of matter that allow the storage and manipulation of more bits, the idea of different computer architectures quickly surfaced. The core of the idea is that strictly adhering to the Church-Turing computational principles is not always rational, especially if these principles do not fit the capabilities of the matter we want the computer to be made of. Maybe it is more rational to make computers that do not qualify as Church-Turing implementations, but are quicker or more energy efficient than their Church-Turing counterparts. One approach for doing this, is to tolerate a certain rate of error in the computation in order to gain dramatic improvements in speed. Certain applications do not need perfectly accurate calculations anyway (for example, in a fast-paced flight simulator, who would notice tiny errors in the movement of the water in

the ocean below the plane?), and in other applications error prevention can be added at higher levels of the architecture (IEEE Spectrum, 2009).[2] Quantum computing also exploits the nature of matter to go beyond the Church-Turing computation. And last, but not least, one might be motivated to investigate non-Church-Turing computation because certain problems are hard to solve on Church-Turing machines. For example, path planning in a labyrinth can be computed by exploiting the diffusion of liquids (Heiko Hamann and Heinz Wörn 2007).

The next chapter discusses computation that can be made on everyday computers. This example will help to explore the nature of embodied computing, which, in turn, will enable us to find answers to some original questions about emergent computing.

3. Evolutionary Computing

There are many applications that run evolutionary computations and genetic algorithms. One such application is *Tierra*, created by the biologist Thomas Ray. He defined a virtual computer architecture that used a 5-bit instruction set, meaning that there were 32 different instructions.

For this architecture, he wrote a program that was able to copy itself repeatedly to different memory locations. In this way, the copies of the small program slowly consumed the available memory space. However, sometimes the system modified a random bit during its operation. This represents a mutation. Most of the time the modified bit makes the program dis-functional. But sometimes the modified code is functional, and rarely it is even more viable than the original (e.g., the program is made of less instructions but has

[2] http://spectrum.ieee.org/semiconductors/processors/cpu-heal-thyself/4

the same behavior). In other words, a process of artificial evolution starts (Ray 1991). A similar experiment is *Nanopond*[3], which uses a 4-bit instruction set and contains no initial program – the initial program itself is created by modifying random bits in a huge memory space.[4]

4. Evolutionary *Game of Life*

For the sake of a simpler analysis, instead a 4- or 5-bit architecture, let us investigate the much simpler system of the *Game of Life*.

"Life" is John Conway's classic "recreational" mathematical game, which became widely known as *Game of Life* because of a *Nature* article (Gardner 1970). The game is played in a 2 dimensional grid, like a sheet in a math exercise book. Cells are marked with two different colors, one for "live" cells, the other for the "dead" cells[5]. The gameplay is divided to "generations," according the following simple rules:

1. **Survival.** Every living cell that has two or three living neighbours survives.
2. **Death.** Every cell that has four or more living neighbours dies ("over-population"). Every cell that has one or zero neighbours dies ("isolation").
3. **Birth.** Every dead cell with exactly three neighbours becomes alive.

Figure 1 shows how the game works[6].

---

[3] http://adam.ierymenko.name/nanopond.shtml

[4] For an 8 hour long Nanopond run, see:
http://video.google.com/videoplay?docid=4775386042852459808

[5] Conway had other rules to limit the initial patterns that are irrelevant to us.

[6] The patterns are the same as the ones published in the original *Scientific American* paper.

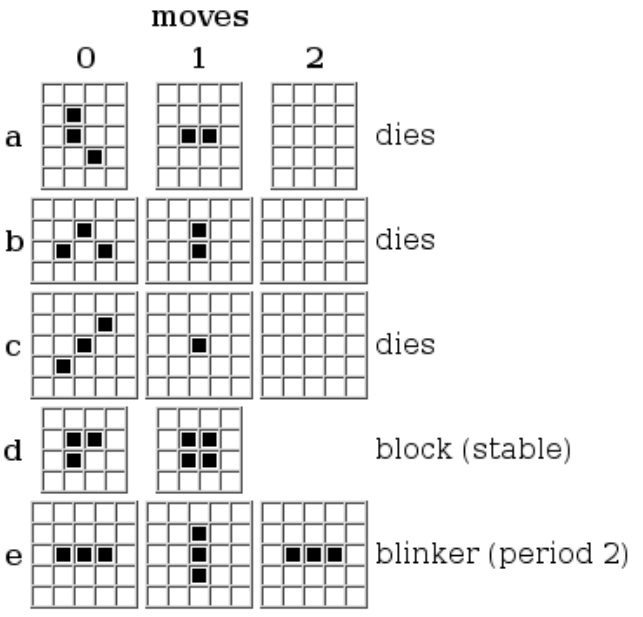

Figure 1. Game of Life patterns

Let us run a genetic algorithm on this universe in a thought experiment![7] Let the goal be the creation of "interesting" *Game of Life* patterns. We will try to invoke evolution to solve the problem: we generate a number of random patterns, and then we select the fittest to achieve our goal. On the patterns selected, we execute *genetic operators*: mutation and crossing. This way the next generation is created, with which we will start the whole process again.

Let us define "interesting" patterns as the ones that are able to maintain the following properties for the largest number of generations:

1. They do not exceed an *nxk* size rectangle.
2. They do not die completely.
3. They do not stabilize (get "frozen").
4. They do not repeat previous properties, or in other words, they do not

[7] Wim Hordijk reports on similar experiments in his PhD dissertation (Hordijk, 1999).

enter infinite loops.

The *fitness* itself will be the number of generations for which a pattern can maintain the properties above. In everyday terms, we are looking for the patterns that are able to transform themselves as long as possible without exceeding the size limits or getting into a loop.

Let us say that after running the experiment for several millions of generations, finally we get a pattern that slowly but steadily grows until reaching the size limits, and then starts to shrink. Let us call this pattern a *pulsar*!

Now, let us try to analyze exactly how a pulsar is created. First, we ran *Game of Life* for billions of generations. This computation, even though it had a physical realization in the computer, was a strict implementation of the abstract algorithm of the game. Therefore, these parts of the calculation were independent of the actual computer we ran it on. They could have been executed by any computer, or even by a group of diligent school kids with math exercise books. Should we run these parts again, we would get the same results.

But the case of the genetic operators is different. Naturally, these too were executed by a computer algorithm. In the case of the mutation, we generated an *(r,t)* pair of coordinates *i* number of times. Then we changed the status of the designated cell from dead to alive or vice versa. In the crossing step, we crossed random parts of two patterns. The algorithms of these two genetic operators, however, are parameterized using random numbers.

So the next question concerns the generation of random numbers. There are many methods to solve this problem. One approach is using Pseudo Random

Number Generators (PRNG). These are mathematically fully defined algorithms, which, using a certain set of initial parameters, can give numbers that approximate the properties of good random numbers: that is, they do not start to give the same numbers again for a long number of generations, and the generated numbers are nicely distributed between a minimum and the maximum value, instead of forming a tight pack around a certain number. One important initial parameter for PRNGs is the "seed" that is used to generate all the random numbers. If one inputs the same seed and the same additional parameters to the same PRNG algorithm in two different instances, the exact same numbers will be generated. The seed is usually provided by a programmer or system administrator, who randomly presses 20-30 keys on the keyboard or enters a favourite poem (which is also converted to a number). In other cases, certain properties of the hardware that are accessible within the system are used, like the serial number of the motherboard. It is also possible to generate a seed from a readable input like the temperature gauge of the processor.

PRNGs have many problems. The biggest one is that with certain (*a priori* unknown) seeds, the period in which the random numbers do not repeat can be surprisingly short. With other seeds, the distribution is uneven. Because of these problems, no current mainstream operating system uses PRNGs on their own. Instead, they use input from the environment of the computer to generate random numbers: the key presses of the user; the network interrupts that are generated by the user and the others communicating with her, e.g. in chat; certain properties of the physical layer of the network (e.g. the variations in strength of the wifi signals). Sometimes the spinning characteristics of the hard drive are used, which depend on the temperature and the position of the device; various temperature gauges that measure the processors, the discs, or the temperature of the palm rest can also be used; audio and video input

devices or even the user's file names or creation times might be used. These inputs fill up an "entropy pool" in the system. Of course, the distribution of such input data can be uneven, but that can be solved in algorithmic ways, e.g. by PRNGs. On the *Linux* operating system, the inner workings of the entropy pool are publicly known. In other operating systems the code is not publicly known, but a similar system is likely to be used.

Additionally, there are dedicated hardware components that are able to generate random numbers by measuring quantum effects such as the breakdown effect of a P-N junction. These solutions are mainly used computer cryptography applications, where large numbers of unpredictable random numbers are required, e.g. in banking servers.

This means that if I run an evolutionary *Game of Life* experiment on my laptop right now, then the keys I press at this moment, as well as the speed of my typing, will affect the random numbers used in the genetic operators. Similarly, the temperature of this room, the angle at which I hold the laptop, the quality of my wifi signal, etc. might also be factors in random number generation. The examination of *Tierra*'s or *Nanopond*'s source code reveals that they do not have their own methods for generating random numbers and rely on the operating system's function made for this purpose – meaning that they also rely on the external factors just described.

As a consequence, we know that Thomas Ray's interesting results were affected by the environment of his computer, as well as the results of the *Nanopond* experiments.

We can conclude that the result of running a genetic algorithm is a product of many physical processes. Some of these processes are just the realizations of

well-defined algorithms, and the particular manner of the physical realization does not affect the result. Other processes involved are, apparently, completely outside the realm of mathematics. These might very well be perfectly deterministic processes on their own, but they should be in a *random relation* to the computer that executes the computation (for the definition of randomness used here see Paksi 2012).

Moreover, these processes are selected exactly on the basis that they are hard to describe mathematically and are contingent on the computer's internal state. This makes it impossible to predict the random numbers, which increases security.

To summarize, the result of the evolutionary computation is a product of many processes, and only a few of them are algorithmically described.

Following this line of thought, we can recognize that not only the special case of evolutionary computation behaves like this, but most of our real life applications. For example, the state of the word processor I'm using right now is a product of its code, but even more of my intentions.

However, the popular view of computing is completely different, and is dominated by the image of an uninterrupted algorithm executed by a Turing machine, with a pre-filled input tape. But the applications that fit this view are exceptional cases: long, time consuming simulations of air or liquid flows around a building; the final rendering of animated movies created by 3d technology; computing the next digit of pi; or finding the next prime number are examples that come to mind. Therefore, cases in which the result of a computation is fully determined by its initial parameters and algorithm are exceptions rather than the rule.

5. Emergent computing

Why is it so important to discuss the embodied nature of computation before analyzing emergent computing? Because, in the popular view of computing, computations are mostly treated as algorithm realizations, and they are thus analyzed by using algorithm theory.

Let us return to the argument discussed in the first part: that the central element of emergent computing is surprise which occurs when engineers create a computation program on level L1, and an observer recognizes a complex phenomenon on a higher level, L2. However, the argument continues, the observation on L2 is nothing more than a set of the elements of L1, which are, in turn, determined by any given state of the program.

But the example of the evolutionary *Game of Life* showed that the pulsar pattern cannot be derived from the algorithm itself. The pattern is the product of the algorithm *and* other physical processes.

Similarly, it is not true that a given state of a computation is always reducible to the previous state, and, through a finite number of steps, to the initial state. Also, a given state of a computation is not deducible from the initial state itself.

If we want to explain the current state of a computation in general, then we have to record and take into account all of the input from physical processes since the initial state. And, what we will see, is that certain physical processes independent of the algorithm will push the computation into a certain direction, for example by creating a pulsar.

Another property that is often mentioned in connection to irreducibility is unpredictability. If we consider a computation that takes initial parameters only and then works without any additional input, we will find that every state

really is deducible from the initial state; as a corollary, every state is predictable – by a faster execution of the same program on a different computer. This also means that the computation in question can have multiple realizations. This is not the case with evolutionary *Game of Life*, *Nanopond*, *Tierra,* or a word processor. Predicting the state of a given future step would involve not only calculating the algorithm faster, but also predicting the states of the physical processes relevant for computing, that is, the processes used in random number generation (or, in the case of the word processor, my sentences). However, it was our intention that the processes in question should be un-predictable.

This does not mean merely that prediction is impossible due to some practical epistemic limitation. Neither the algorithm nor the external physical processes determine the outcome of the computation *per se*, and their relation is random. Unpredictability is only a symptom of indeterminacy in this case.

Let us see what role do certain elements play in the evolutionary *Game of Life*'s computation! The algorithm controls the direction of the computation and sets the conditions that a certain state must fulfill. For example, one requirement might be that the fitness in generation *n* should always be higher than in generation *n-1* (in this case, the less fit will always be left out), which results in a monotonous improvement, but has the danger of getting stuck at a local optimum.

However, on a relatively small 100x100-sized grid, the number of possible patterns is 10^3009 (the number of all particles in the universe is estimated as 10^87), and the algorithm itself is not able to compute which directions are promising. This is why we use the principle of evolution, and we include external processes to generate the required random numbers. These physical processes would, of course, not be able to create the pulsar in themselves either.

The principle of the computation is realized by an algorithm, and the goal of the computation is also expressed as a fitness function that is used by this algorithm. The limits and boundaries of the states the computation can reach are also controlled by the algorithm.

However, the success of the algorithm also requires a different set of (otherwise deterministic) processes that bear a random relation to the algorithm.

The pulsar is thus reducible to the algorithm, to the processes, and to their random relation, and all three elements are required to make this reduction. This means that a holistic view of the nature of computing should be accepted, in which the computation is something more than its components. This is certainly more than epistemic emergence.

The next question is whether all these three components are reducible to the principles of one unified worldview, e.g. physicalism? Naturally, the evolutionary algorithm at work has a similar physical realization to the external processes. But can the algorithm completely described by these processes and nothing more?

The evolutionary algorithm is created by a human. If we want to show that problem solving processes can only be emergent in the epistemic sense, we have to show that the human mind itself (which is the cause of the algorithm and the fitness function that sets its goal) does not transcend the level of physical processes. If we want to show that computational processes can be emergent in the ontological sense, we have to show that the results of computation *do* transcend physical processes. But, in order to be able to show this, we first need to argue that the human mind does transcend physical processes, perhaps by arguing that the biological evolution that led to the human is an emergent process itself.

In other words, a resurgence of the traditional debate over the emergence of

the mind and biological evolution is needed to move forward, which lies outside the scope of this paper. What is important to us here is that inquiries into emergent computing are related to inquiries into emergence in general, and thus, any development in the latter debate will be relevant for a better understanding of emergent computing, since in reality, there is no decisive difference between generic (e.g. biological) and computational processes and behaviour.

6. Conclusion

The aim of this article was to show that emergence in emergent computing is not necessarily to be understood in a merely epistemic sense, as the result of the computation cannot be reduced to the processes that are involved in its production. This is due to the embodied nature of computation. The question of emergent computing is also connected to the question of the emergence in general through the human mind which develops algorithms and sets goals for computing

However, the inquiry into the emergence of the human mind and emergent computing could work in various directions. By experimenting with emergent computing applications like machine evolution, we might gain a better understanding of the questions of emergence in biological evolution and the human mind and vise versa. What this paper argues for is that both classes of these processes share the feature of being embodied with all of the consequences.

References

Brown, J. R. 2008. *Philosophy of Mathematics: A Contemporary Introduction to the World of Proofs and Pictures*, London: Routledge.

Callaghan, A. 2006. Emergent properties of the human immune response to hiv infection: Results from multi-agent computer simulations. *ERCIM News*, 64:48–49.

Cariani, P. 1991. Emergence and artificial life. *Artificial Life II, SFI Studies in the Sciences of Complexity*, pages 775–796.

Cooper, B. S. 2011. From Descartes to Turing: The Computational Content Of Supervenience. In *Information and Computation* (editors Mark Burgin and Gordana Dodig-Crnkovic), World Scientific Publishing Co., 2011, pp.107-148.

Curtis, S. A. et al. 2003. *Ants for human exploration and development of space*. volume 1.

Forrest, S. 1990. Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *Physica D*, 42:1–11.

Gardner, M. 1970. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223:120–123.

Hordijk, W. 1999. *Dynamics, emergent computation,and evolution in cellular automata*. PhD Dissertation.

Paksi D. 2011. Emergence and Reduction in the Philosophy of Michael Polanyi. Part II. APPRAISAL (LOUGHBOROUGH) 8(4) pp. 28-42. (2011)

Paksi D. 2015. The Meaning of Randomness: The Laplacian Fault of neo-

Darwinians According to Michael Polanyi. Appraisal. 10(3) (2015)

Ruskin, H. and Walshe, R. 2006. Introduction to the special theme: Emergent computing. *ERCIM News*, 64:24–26.

Sipper, E. M. and Giacobini, M. 2008. Evolutionary computation in games. *Genetic Programming and Evolvable Machines*, 9(4).

Walshe, R., Ruskin, H. J., and Callaghan, A. 2008. Multi-agent simulations of the immune response to hiv during the acute stage of infection. *International Journal of Modern Physics C*, 19(2):15–32.