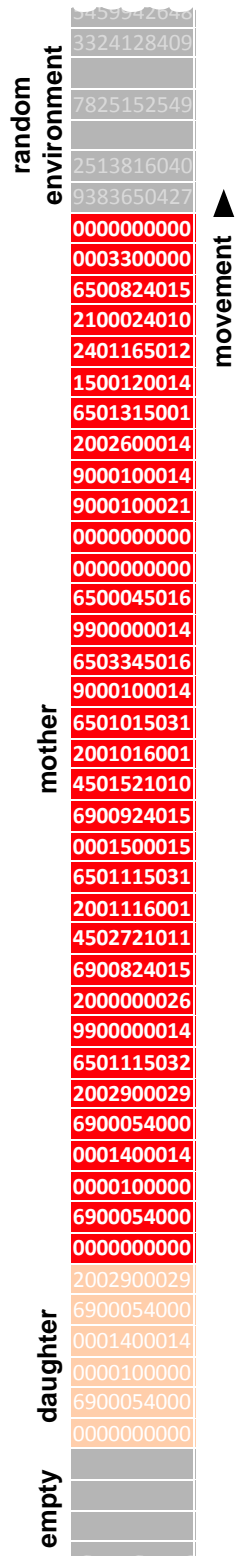


1960 – the first artificial universe

complex artificial life in a
Darwinian world

Frederick G. Stahl

March 2013
fred@fredstahl.com
www.fredstahl.com





1960 – the first artificial universe

complex artificial life in a Darwinian world

Frederick G. Stahl

March 2013
fred@fredstahl.com
www.fredstahl.com

the first operational artificial universe

complex artificial life in a Darwinian world

In 1960, I created an artificial universe inside a digital computer. It had numeric creatures that moved through the universe eating numbers. If they found another creature, they would eat it. They could reproduce and did. Mutation was possible and did happen. And the creatures could observe the world around them and could calculate and make decisions based on what they saw, and they did. It was the first operational artificial universe with complex artificial life.

the first design

John von Neumann's brilliant mind is a 20th century legend. Beginning in 1922, at the age 19, until his death in 1957, he made significant theoretical contributions to mathematics, quantum physics, computer science, mathematical economics, and more. In 1931 he became a member of the faculty of Princeton University. Two years later he was among the first six professors of mathematics appointed to the newly formed Institute for Advanced Study in Princeton, a position he held until the end of his life.

Von Neumann's interest in theoretical hydrodynamics drew him into the American war effort and eventually into the Manhattan Project and the development of the atomic bomb. Numerical methods for solving the differential equations describing the interactions of shock waves in nuclear detonations required enormous calculations. Von Neumann helped the bomb builders improve computational efficiencies, which in turn involved him with early computers. He was a consultant to a university team that in 1945 came up with the revolutionary computer architecture that placed both data and instructions in the same addressable high-speed memory, an architecture that became the standard.

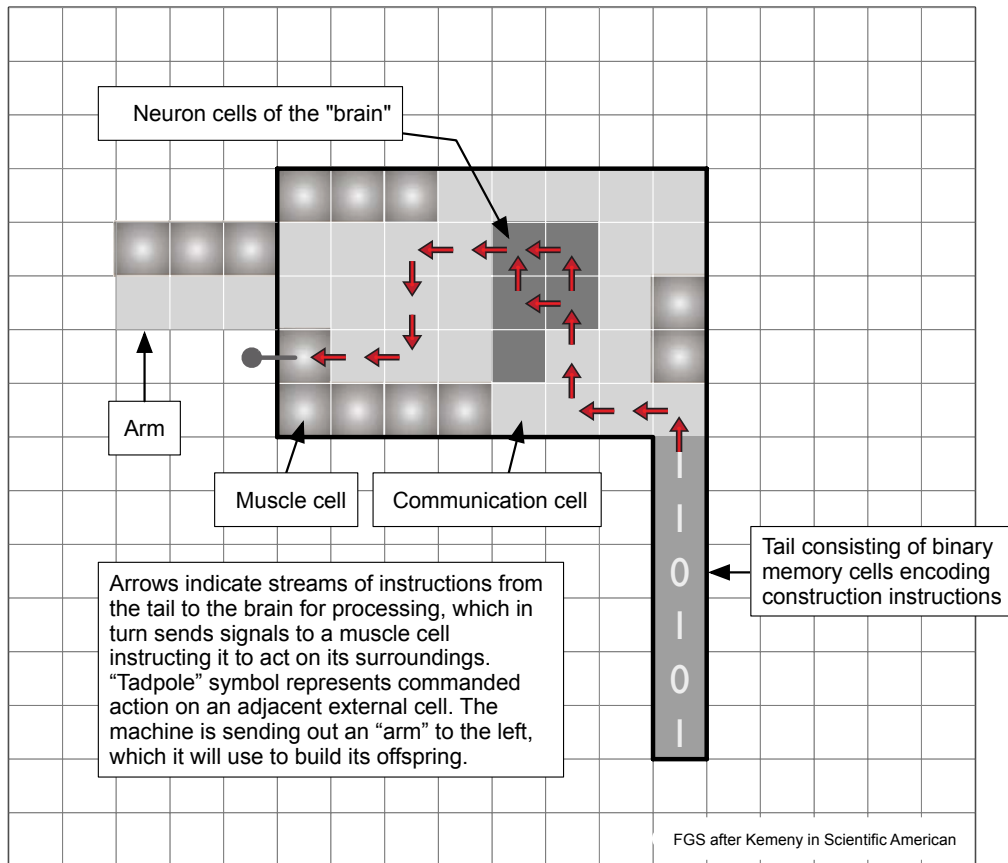
At about the same time, von Neumann became interested in the theoretical limits of self-reproducing machines compared to the human brain. To make the problem theoretically tractable, von Neumann discarded the paradigm of a physical machine in a field of parts in favor of a conceptual machine in a cellular world. He lectured about self-reproducing automata in the late 1940s and early '50s. While some notes from von Neumann's lectures had circulated privately, none appeared in print until 1955. It was then that *Scientific American* published an article written by John Kemeny that summarized his 1951 lecture notes taken when he was new faculty member at Princeton.¹

According to Kemeny, von Neumann explored the question *what is the essential difference between a man and a machine?* by conceiving of a self-reproducing machine. The machine in von Neumann's conceptual design assembles raw material from its environs to build another machine. Each machine contains a digital code that directs the fabrication process and thereby implicitly specifies its design. After completing assembly of an offspring machine, the mother copies the construction instructions from herself into her daughter and activates the offspring. Duplication of the construction instructions into the offspring is analogous to inheriting genetic material from a parent in the natural world.

Under control of its own code, the daughter then searches for “food” and builds another machine guided by the construction instruction specified in its inherited “genetic” code.

Could such machines evolve? Von Neumann noted that random changes introduced during the process of copying the genetic code from the mother to the daughter are like mutations. The grandchildren are built according to, and carry, the mutated code. By design, a component of one machine looks like raw material to another machine. A machine could “eat,” and thereby kill, another machine. Consequently, limiting available resources in the environment would increase competition among the machines and intensify evolutionary pressures.

Von Neumann embedded his machines in an unbounded, two-dimensional space of cubical cells, each of which, if not empty, contains either raw material or a cell of another machine. There are four classes of cells. *Muscle cells* effect changes in the external environment. *Neuron cells* are organized to read construction instructions and transmit control orders to muscle cells through *communication cells*. In von Neumann’s blueprint, the basic body of the machine consists of 32,000 cells in a rectangular box 80 by 400 squares. The genetic code resides in a linear tail of *memory cells* attached to the body. The tail is one cell wide and 150,000 cells long. It can be thought of as a rigid binary tape that can be read by the structure of neurons. The following graphic shows the four kinds of cells in a notional diagram of von Neumann’s machine.



Notional diagram of the arrangement of cells in von Neumann's machine

Not until 1966 was a comprehensive collection of von Neumann's work on the theory of self-reproducing automata edited and completed by Arthur W. Burks.² Burks, who had worked with von Neumann in 1946, writes that von Neumann's interest in the complexities of self-reproducing machines was the first step in his search for a theory of automata that would lead to the design of extraordinarily powerful computers based on logic that will "strongly resemble and interconnect with probability theory, thermodynamics, and information theory." Tragically, von Neumann's untimely death at the age of 53 ended his journey of discovery. Part of the material Burks assembled into the book was written by von Neumann while in the hospital before his death.

creating an operational universe in a computer

I had an early interest in computer architecture. In 1954, at the age of 15, I wrote a paper describing my original (though hardly new) designs using electro-mechanical relays to do binary arithmetic. It won third place in the Michigan Westinghouse Science Contest. I first learned of von Neumann in early 1955, when I wrote a high school essay on game theory. It was only when I read Kemeny's article in *Scientific American* in the summer of that year that I understood von Neumann's central role in the emerging science of digital computers. But the idea of trying to functionally simulate von Neumann's machines with a digital computer did not occur to me then.

In the fall of 1956, the beginning of my second undergraduate year as a math major at Wayne State University, I began working as a part-time programmer at the university's Computation Laboratory to pay my college expenses. The Lab had two vacuum-tube computers: a new IBM Type 650 and the UDEC, an earlier, one-of-a-kind machine built by Burroughs. When the computers were idle, we students working at the Lab could use them for personal research projects. I had a interest in artificial intelligence. I built a program that learned how to play the game of Hex by watching the moves of its human opponents.

Only after I returned to the Lab after a year at the University of Munich did the idea of a digital simulation of von Neumann's creatures occur to me. In the fall of 1959, just before I began my first year of graduate school, I read a piece on self-reproducing machines written by Lionel Penrose, a British psychiatrist, medical geneticist, and mathematician (and father of Roger Penrose, the now-renowned English mathematical physicist). In an otherwise unremarkable article in the June, 1959 issue of *Scientific American*, the senior Penrose included a paragraph summarizing von Neumann's concept of a self-replicating machine:

The [general] theory [of self-reproduction] has two aspects, which can be called the logical and mechanical. The logical part was first investigated by late John von Neumann of the Institute for Advanced Study in Princeton, N. J. He decided in 1951 that it must be possible to build an engine that would have the property of self-reproduction. The method would be to construct a machine that is capable of building any describable machine. It would follow logically that such a machine would be able to build another machine just like itself. Each machine would carry a sort of tail bearing a code describing how to make the body of the machine and also how to reprint the code. . . . The machine would assemble these parts from raw material in its environment, organize them and transform them into a new replica of itself.³

I immediately thought of the 1955 piece by Kemeny and the graphic of the notional von Neumann machine much like that presented above. I envisaged a *digital* simulation of an extended concept of von Neumann's notional machine. If I could make my creatures mobile in a digital universe with others of its species then I might have lethal competition. If, as von Neumann had conceived, I included digital mutation in reproduction and if the digital entities could kill and eat each other then I would have survival of the fittest. "Life" in the universe would be Darwinian. With luck I might even observe a little evolution.

I realized immediately that my vision was completely impractical. Simulation of even one of von Neumann's machines was clearly beyond the capabilities of the computers at hand. The main memory of the larger of the two machines at the Lab, the IBM 650, had only 2,000 "words" of ten-digits each. If each digit represented one cell then only 20,000 cells were available, even with no allowance for the memory needed to host the supervisory code. Complexities of von Neumann's automata, with its ten and hundreds of thousands of components, were orders of magnitude beyond available memories of computers of the day, let alone the modest capacity of the 650.

My strongest memory from that time was deep disappointment. The structure of digital creatures in any universe I might be able to build would have to be radically different than von Neumann's. That picture of von Neumann's boxy machine with an arm and a tape memory has always remained with me—so conceptually close but then technically so far in the future.⁴

In thinking about an alternative approach, I hit upon the concept of computer programs as a "living" entities. Its components would be the complete machine-level instruction set of the IBM machine, with the addition of a couple of extra instructions for mutation, giving birth (activation), and locomotion. While such creatures would have no analog in the physical world, I knew that they could, at least in principle, be able compute anything. Since each inhabitant was a general-purpose computer, and since any general-purpose computer can emulate a Turing machine, and since a Turing machine can compute any algorithm then my creatures could compute any algorithm.

Genetic codes presented another problem. Designs, encoded as construction instructions, are a central feature of John von Neumann's machines. The digital blueprints that control construction of offspring and are copied into each daughter as part of the reproduction process. I saw no way in my design of making a compact genetic code integral to one of my digital entities. In a purely digital universe, a machine's genetic design would be too bulky since it could not have fewer degrees of freedom than the digital entity itself.

My solution was to use the digital instantiation of the mother to guide construction of its daughter. In other words, reproduction would be simply making a copy of the mother creature's machine language code. I could still simulate simple mutation. An external parameter m would control rate of mutation using random numbers. During reproduction, each decimal digit of the daughter entity had an average rate one per ten to the power m of being replaced by a random digit rather than by the value of the respective digit of its mother. Each creature in my universe could in principle construct a creature of any design. I limited replication to self-reproduction.

implementation

As built, the universe was 1,350 ten-digit words logically connected in a circle. Each entity consisted of sixty machine language instruction occupying 32 words of memory. See the appendix of my 1961 paper (following) for the detailed operational design of the first creature. There too will be found the extensions of the IBM 650's native numeric operation codes added to operate the universe.

The Right Hand of God, the supervisory program for the physics of the universe, occupied 650 words. It moved time forward by sequentially executing instructions in the computer program of each "living" entity. When more than one entity was alive, the Right Hand of God would time-share execution of their programs. Like a primitive multi-tasking operating system, the control program would cycle through all living creatures, executing one instruction from each entity in each cycle.

The IBM Type 650 had no operating system and no provision for multitasking—like hardware interrupts or software breakpoints. To retain control over the execution of an entity's code, the Right Hand of God executed all creature code interpretatively. I believe that the Right Hand of God was among the first multi-tasking operating system.

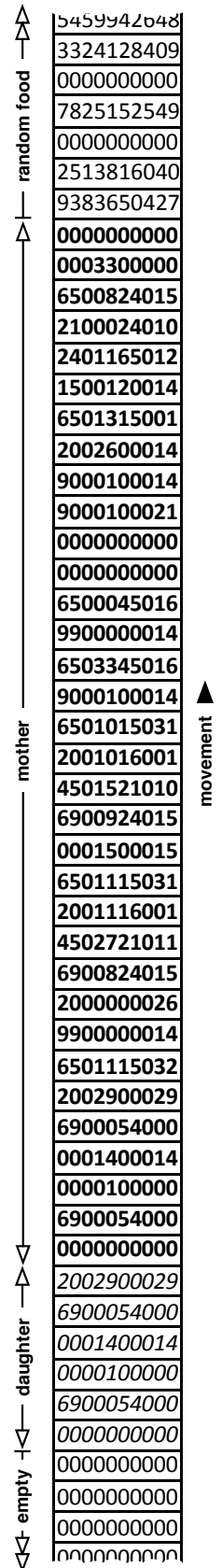
In operation, the digital creatures moved themselves through this circular linear universe consuming food (non-zero memory cells) leaving behind them empty cells. (Digital creatures were indistinguishable from food.) When an entity had accumulated a quantity of food equal to its size, it would begin to build a copy of itself as it crawled forward. Each time the creature moved forward one space, it would put a copy (possibly with mutations) of one word of its program in the space just vacated behind it. When replication was complete, the mother would notify the Right Hand of God that her daughter was ready to begin living as an independent creature.

The graphic to the right show a mother (bold) creating a daughter (italics). With every step the mother leaves behind another cell of the daughter, a total of six so far in this graphic. The aborning daughter is stationary until replication is complete. The spaces behind the daughter are empty since the mother ate any food that was there.

what happened?

In February of 1960 I completed debugging the software and creature code and made the first and only production run of my artificial universe—with disappointing results (see "On Artificial Universes" following).

A shortly after that, I brought my work to the attention of Prof. Walter Hoffman, head of the Computation Laboratory. He arranged for us to meet over a college cafeteria lunch with one of the younger and more visionary professors in the Biology Department. We explained the experiment. We talked about its innovations and its limitations. We asked him about his interest in working with us. He was decidedly uninterested.



I set the project aside. I put the deck of IBM punch cards containing the software for the artificial universe on a high shelf in my office and turned to other efforts.

the first artificial universe?

In 1953, Nils Aall Barricelli simulated the emergence of simple, life-like genetic structures from inert particles using an electronic computer at the Princeton Institute for Advanced Study. His goal was to study whether symbiotic association of simple particles could evolve into improbably complex structures, much as living, self-reproducing cells with genetic codes might have emerged from a soup of organic compounds in the real world.⁵

Barricelli's computer world was 512 bytes of machine memory organized into a flat matrix (imagine 16x32) of cubicles. The software that operated the universe connected each edge of the matrix of cubicles to the opposite edge, making the world topologically finite but unbounded. Each cubicle contained either a zero, which meant empty, or was occupied by one of 15 different particles (or "genes") identified by a number between 1 and 15. Each particle in Barricelli's universe had only two characteristics: its type (one of the 15) and its location.

The operating logic and the "physics" of his world were encoded into simple external rules of engagement. The operating software in Barricelli's world would use the rules to determine dynamically the results of symbiotic and parasitic interactions among particles. To begin a run, Barricelli strewed some random numbers into the cubicles. Then, the operating software moved time forward in computational cycles or "generations." In each generation, the operating software applied the rules of engagement to compute the location of each gene in the next generation. For example, a gene could shift to a new location only if certain other kinds of genes were adjacent. Other rules specified the outcome of conflicts, as when two genes were directed to the same cubicle. Reproduction rules allowed replication of a gene only when genes of certain kinds were present. Certain collisions among groups of genes produced mutations.

Barricelli's experiments involved thousands of generations, some in larger matrices of cubicles. After some adjustments to the rules, certain combinations of genes achieved persistent reproductive success and came to dominate Barricelli's world. But evolution always stalled after achieving only modest levels of complexity. Perhaps more complex rules and maybe more kinds of particles would have released evolution to higher levels of complexity.

Barricelli's achievement was conceiving and operating a universe in which elementary non-reproducing particles evolved into "symbioorganisms," as he called them, that under some set of rules were self-reproducing structures. He held that the inhabitants of his universe were not simulacra; they *were* the "life-like" structures.

Barricelli's is a microscopic cellular world. Mine, like von Neumann's, is macroscopic. Particles versus machines. Von Neumann started with a requirement that the machines be "constructively universal," which means they can in principle build any computing machine. That criterion is present only to guarantee that his machines can self-reproduce. The machines in my universe can and did build copies of themselves, and in principle can do so no matter how complex they are. They are therefore also "universal machines." Consequently, my machines had all the essential features and potential complexity von Neumann required of the machines. Here is how Bruce Damer, a leading researcher of artificial life, compares Barricelli's and my universes:

Fred Stahl implemented one of the first von Neumann inspired self-reproducing cellular automata (CA) systems on an IBM 650 mainframe. . . . Stahl's universe went beyond Barricelli's in terms of complexity. . . . [I]t featured an implementation . . . of a universal machine . . . for each of his simulated creatures using the computer's instruction set. . . . [It] featured food and competing creatures capable of reproducing and mutation.⁶

I believe therefore that my 1960 project was the *first ever operational artificial universe with complex artificial life in a Darwinian world*.

the first computer virus?

Some people believe that the code for my creatures is the first computer virus. The following appears in the archeology section of www.securelist.com, a website specializing in Internet security:⁷

Historians are still debating when the first computer virus really appeared. We do know a few things for certain, however: the first computer, which is generally considered to have been invented by Charles Babbage, did not have any viruses. By the mid-1970s, Univac 1108 and IBM 360/370 did.

Nevertheless, the idea for computer viruses actually appeared much earlier. Many consider the starting point to be the work of John von Neumann in his studies on self-reproducing mathematical automata, famous in the 1940s. By 1951, Neumann had already proposed methods for demonstrating how to create such automata.

In 1959, the British mathematician Lionel Penrose presented [von Neumann's] view on automated self-replication in his Scientific American article 'Self-Reproducing Machines'. . . . Penrose described a simple two-dimensional model of this structure that could be activated, multiply, mutate and attack. Shortly after Penrose's article appeared, Frederick G. Stahl reproduced this model in machine code on an IBM 650.

It should be noted that these studies were never intended to providing a basis for the future development of computer viruses. On the contrary, these scientists were striving to perfect this world and make it more suitable for human life. And it was these works that laid the foundation for many later studies on robotics and artificial intelligence.

The final sentence puts my work in proper perspective.

my 1961 paper describing the experiment

In early 1961 I wrote a description of my artificial universe and its artificial life. Entitled "On Artificial Universes," it describes the concept, design, and operation of the universe and its creatures. It gives details of the results of the production run. It presents my conclusions and wraps up with what I thought were relevant philosophical and moral issues.

“On Artificial Universes” went unpublished. A short summary of my digital universe and its artificial life did however appear in print in 1985. A year earlier, A.K. Dewdney described battling computer programs in his “Computer Recreations” column in *Scientific American*. I responded with a letter to him describing my 1960 artificial universe project. I wrote that my universe enabled reproduction with mutation, unlike the software entities his earlier column discussed. With my 1984 letter I enclosed a copy of my 1961 paper. Dewdney published a brief description of my work in his column in the March 1985 issue of *Scientific American*.

My copies of “On Artificial Universes” were lost decades ago. Or so I thought until recently when I discovered a paper copy. It is reproduced on the following pages.

additional material in this package

“On artificial Universes.” Unpublished paper by Frederick G. Stahl. 1961.

Annex A – Description of my artificial universe in *Scientific American* 1985.

Annex B – Digital Computers at the Wayne State University Computation Laboratory 1960.

notes

¹ John G. Kemeny. “Man Viewed As a Machine.” *Scientific American* April 1955: 58-67.

² John von Neumann. *Theory of Self-Reproducing Automata*. Edited and completed by Arthur W. Burks. (University of Illinois Press, 1966).

³ Lionel S. Penrose. “Self-Reproducing Machines.” *Scientific American*. June 1959: 105–114.

⁴ Renato Nobili and Umberto Pesavento. “An Implementation of von Neumann’s Self-Reproducing Machine,” *Artificial Life* 2 (1995) no. 4: 337–354.

⁵ The description of Barricelli’s work draw’s on a presentation by George Dyson: “Darwin Among the Machines; or, the Origins of Artificial Life,” sponsored by Edge, July 8, 1997. (www.edge.org)

⁶ Bruce F. Damer. *The EVOGRID: An Approach to Computational Origins of Life Endeavours*. Ph.D. thesis, University College Dublin, May 2011. P. 30.

⁷ <http://www.securelist.com/en/threats/detect?chapter=105>.

On Artifical Universes

by

Frederick G. Stahl

1961

Paper was written in 1961. In 1984, the paper was copyedited to correct grammatical, syntactical, and typographical errors before submission to *Scientific American*.

Following is a scan of the typewritten 1984 version.

ON ARTIFICIAL UNIVERSES

by Frederick G. Stahl

In February of 1960, I conceived, built and set into operation my personal universe. To inhabit my universe I created some crude little creatures that lived, ate, reproduced, killed and died according to a set of specially designed physical laws. Reproduction was imperfect; that is, mutation was allowed. Evolution was therefore theoretically possible in this universe. The creatures were cannibalistic because they could not in their initial form distinguish whether material (food) was or was not part of the body of another creature. Thus were the creatures subject to selective pressure.

This paper discusses the details of this little universe, its creatures, and their collective fate. The paper closes with some general evaluative and philosophical remarks.

THE UNIVERSE

The universe and the creatures therein were realized in the memory of a digital computer. The memory was divided into two parts: that occupied by the universe and that occupied by the program called the Right Hand of God that operated the universe.

The universe was quantized into 1350 spaces. Each space was equivalent to one word of memory. The IBM Type 650 was used in which each word of memory is ten decimal digits and a sign. The spaces were arranged in a closed, linear fashion making the universe topologically equivalent to a circle.

Physical matter was defined as a nonzero space; that is, if the computer word had at least one nonzero digit, then the entire space was considered occupied by a chunk of matter.

Energy had no representation in the universe.

THE CREATURES

A creature was initially a set of pieces of matter occupying 32 sequential spaces represented by 32 ten digit numbers. These numbers were coded to represent the detailed life functions of the creature.

It was these codes composing each creature which were interpreted by the Right Hand of God program that operated the universe. In a unit of time, the Right Hand of God program would sequentially interpret a fixed number of those codes and execute the corresponding functions. As such, the Right Hand of God program embodied the physics of the universe.

The codes of the original creature were programmed by me to cause the creature to execute the following functions:

- o Lurch forward one space
- o If the space directly in front of the creature contained matter (was nonzero), then increment the count of units of matter encountered before movement.
- o If the creature had accumulated enough pieces of matter to equal its size, it would iteratively copy one space of itself onto the space directly behind itself before every movement.

Unless reproducing, the creature would leave empty spaces behind it. Matter in spaces before the creature disappeared as the creature successively occupied those spaces.

The effect was of a creature crawling along, looking for and accumulating food until it had saved enough to construct an offspring. Then as it vacated spaces while crawling, a copy of itself would be left behind. Meanwhile, the creature would be accumulating more food for yet another offspring.

Once copying of an offspring was complete, the creature would start the offspring living, which was done by notifying the Right Hand of God program of the offspring's location. The Right Hand of God program would then have another creature to operate, more or less simultaneously with any other.

Copying of an offspring creature had two important features:

- o The copy was direct. It was not based on a description of the creature (as von Neumann conceived of self-reproducing automata) but rather the copy was a space-by-space duplication of the actual structure of the parent.
- o The copy was imperfect. Random variations were introduced in the digits of the copy at the average rate of one per ten to the power X , where X was an input number.

The first feature implies that acquired as well as inherited characteristics are passed on to offspring. The second feature provides for mutations. Combined with a competitive environment, such genetic shifts combined with survival of the fittest should in theory lead to development of more efficient or vicious strains.

Once an offspring is complete and it has been registered as a birth, the Right Hand of God program delays initiation of its life. This wait gives the parent time to crawl away. Otherwise the offspring

might gobble up its parent. This is a possibility because a feeding creature crawls more slowly than one not eating. The average rates are:

Activity -----	Spaces Crawled per Unit Time -----
Nothing	4
Eating	2
Reproducing	1 1/3
Eating and Reproducing	1

The parent is running into random food; the offspring finds nothing ahead of its since the parent leaves no food behind it, at least for a while after the birth of the offspring.

THE RIGHT HAND OF GOD PROGRAM

The principal functions of the Right Hand of God program are apparent from the preceding discussion. Death decision has not been touched on yet.

The Right Hand of God program operates by executing interpretively computer-like code which compose each creature. The functions used to define the creature are essentially equivalent to the operation codes of the host computer augmented by an imperfect STORE operation and a special birth operation. Otherwise almost all of the arithmetic, logical, control and input-output (via punch cards) operations are included. (The table look-up function is excluded as are a couple of other redundant operations.) While the IBM Type 650 is a two address machine, the pseudocode used for the creatures is a single address system. All addresses are relative to the base address of the creature. The appendix shows the pseudocode program for the initial creature.

With that background, the death criteria can be defined:

- o In executing the pseudocode of a creature, the Right Hand of God program encounters a zero.
- o In executing the pseudocode of a creature, the Right Hand of God program encounters an operation code that is undefined.

If the Right Hand of God program detects either of these conditions, no further operation codes are interpreted for that creature--it is left to be eaten by others. A creature X takes a bite of creature Y by moving itself space by space over the tail of Y. The nose and tail of every creature are bounded by a zero space, a potential danger because an attempt to interpret that zero would lead to death. Therefore, if

X bites Y, then one of the above conditions may result in Y's death, X may be ripped in half by Y's movement, causing Y to die, or both may die.

The Right Hand of God program recognizes a larger set of operation codes than the creatures use. In fact, a subset of the creature pseudocodes is equivalent to the order set of the IBM Type 650. An implication of this is that there is no reason to rule out evolution of a creature to do any function a general purpose digital computer can be programmed to do. If computers are capable of thinking then these creatures can potentially evolve into thinking entities, given enough time and space. Whether or not such evolution would take place, even if there were sufficient time and space, is another question.

The Right Hand of God program performs a number of simple bookkeeping tasks such as keeping track of where all the living creatures are in the universe and issuing birth and death notices to the world outside the computer. It also puts out data on crawling progress and whether each creature is eating or reproducing. [The output of the 650 was solely punched cards which were listed offline on a IBM Type 407 tabulating machine.]

FATE OF THE UNIVERSE

Initially the universe was filled with material randomly dispersed with a average density of 0.5. One creature was placed in the milieu and the Right Hand of God program was started. Here's what happened.

Creature #1 started crawling and eating.

Soon came the announcement of the birth of Creature #2.

After a while, the reports showed that both creatures were crawling and eating. That meant that if Creature #2 was a mutant, it was a viable mutant.

Soon Creature #2 was eating the half-formed progeny of Creature #1.

Finally, the obituary for Creature #1 came. Creature #2 had taken a lethal bite of Creature #1.

Creature #2 was the only living entity, but it never did anything but eat and crawl--it was a viable but sterile mutant!

POST-MORTEM

By the time the codes for the Right Hand of God program and the creature were debugged and tested, enough of the faults of the universe were apparent to make further runs pointless. Among the obvious improvements needed were:

- o A larger universe. The memory size of the IBM Type 650 was 2000 words. The Right Hand of God program occupied 650, leaving 1350 for the universe. It is doubtful that too much evolution could take place in such a restricted space. Ideally a space larger by a factor of ten to the twentieth or thirtieth would be preferred, but even a universe on a scale of tens of thousands of spaces might be interesting. At the same time a two dimensional universe might relieve the intensity of the immediate competition and allow more ingenious solutions to the problem of survival. A chimpanzee in a telephone booth with a tiger will not evolve. The universe modeled in this experiment was a telephone booth. A forest is needed.
- o More creatures initially. That would immediately provide a larger population base and therefore greater probability of developing a large, viable, dynamic population.
- o A larger, faster computer. Measures of the basic operation speeds of the IBM Type 650 are in terms of milliseconds. The single run of the universe reported here required approximately one and a half hours. On a modern machine, only six seconds would be required for an equivalent run. Nevertheless, much more time--probably orders of magnitude more--might be required for the long term effects of natural selection and hence interesting evolution to become apparent.
- o Optimum mutation rate. Three key attributes of descendent creatures are determined by their structure: whether each is mutated, viable, or sterile. The exact structure of the original creature is known so that the expected proportions of offspring possessing each of the eight combinations of these binary attributes can be closely estimated as a function of the mutation rate. An optimum rate is defined as that which produces the most viable, mutated and non-sterile offspring in the first generation. For the design of creatures used here, that rate is about two digits per thousand. In other words the probability of replacing any one digit being copied with a randomly drawn digit is 0.002.

- o Conservation of matter. This law should be enforced to prevent creatures from destroying or creating matter. This change would entail drastic restructuring of the physics of the universe and would appear to move the basic operations away from using almost a computer operation code set to something more like physical functions.

OTHER SYSTEMS

At least two other systems of self-reproducing entities have been designed. In American Scientist ("On Models of Reproduction", date and issue identification unknown), Homer Jacobson reported on a hardware system he built and operated. He used toy trains and railroad track. Two kinds of train cars were the basic building blocks, and an entity was a pair--one each of the two kinds. One entity placed on the track would organize uncoupled basic units on adjacent tracks into pairs constituting new entities. While ingeniously done, the system was genetically static. Further, there was no lethal competition among entities.

John von Neumann produced a design for reproducing automata. While only limited information is available [as of the original writing of this paper], it appears that after placing an initial creature in a two-dimensional universe containing unorganized material, the creature will move about selecting appropriate pieces of equipment which it organizes into a replica of itself according to a set of instructions (an analog to a genetic code) which is stored internally. While the creatures move only in a two-dimensional surface, they are essentially three-dimensional in that their functions make direct use of the physical laws of our universe. The genetic code could, for example, be a tape recording.

Von Neumann's creatures are more sophisticated than mine because they have distinct genetic coding. It is not clear how von Neumann's creatures can do much sophisticated information processing which is the potential of my creatures. These differences reflect the underlying purposes. Von Neumann sought to prove by demonstration that artificial but self-reproducing automata were possible. My objective was to try some evolution.

AN APPLICATION

[This section is omitted. In the original paper, it suggested evolution in specialized, problem-oriented environments to develop designs for intelligent automata.]

PHILOSOPHICAL REMARKS

This experiment with an artificial universe provides an interesting framework for thinking about certain problems.

By common useage of the word, I am the god of such an artificial universe as I may create and set into motion. Consider:

- o Its creation was my achievement.
- o Continued existence of the universe is dependent on my will.
- o Relative to the artificial universe, I would be omniscient and omnipotent, for I could stop the universe at any time, examine any part of it in detail, change any part of it, and restart it. If the universe was large enough and evolution had produced intelligence to a degree that the concept of an observer inside the universe would make any sense, then the consequences of the above sequence would transpire instantaneously. I can therefore perform miracles at will.

Such concepts put an interesting light on some theological questions. How do we communicate with the inhabitants of such a universe, if we should want to? Can they, the inhabitants of the universe, conceive of our existence? Not too likely. And we could not demonstrate directly our existence because we could not get into their universe. Nor can they leave theirs to operate in ours.

If we do in fact create a universe that has the potential for evolving into intelligent life, what are our moral obligations? Once created what are our responsibilities to such a universe? Is it entitled to continued existence or can we pull the plug on it without guilt? Consider if apparently intelligent entities evolved in such a digital universe. What rights do they have according our moral laws? Do they even exist?

APPENDIX
CODE OF THE INITIAL CREATURE

CREATURE CODE

Relative Location	Code			
-----	Op	Addr	Op	Addr
-----	-----	-----	-----	-----
0000				0000000000
0001				0003300000
0002	65	008	24	015
0003	21	000	24	010
0004	24	011	65	012
0005	15	001	20	014
0006	65	013	15	001
0007	20	026	00	014
0008	90	001	00	014
0009	90	001	00	021
0010				0000000000
0011				0000000000
0012	65	000	45	016
0013	99	000	00	014
0014	65	033	45	016
0015	90	001	00	014
0016	65	010	15	031
0017	20	010	16	001
0018	45	015	21	010
0019	69	009	24	015
0020	00	015	00	015
0021	65	011	15	031
0022	20	011	16	001
0023	45	027	21	011
0024	69	008	24	015
0025	20	000	00	026
0026	99	000	00	014
0027	65	011	15	032
0028	20	029	00	029
0029	69	000	54	000
0030	00	014	00	014
0031	00	001	00	000
0032	69	000	54	000
0033				0000000000

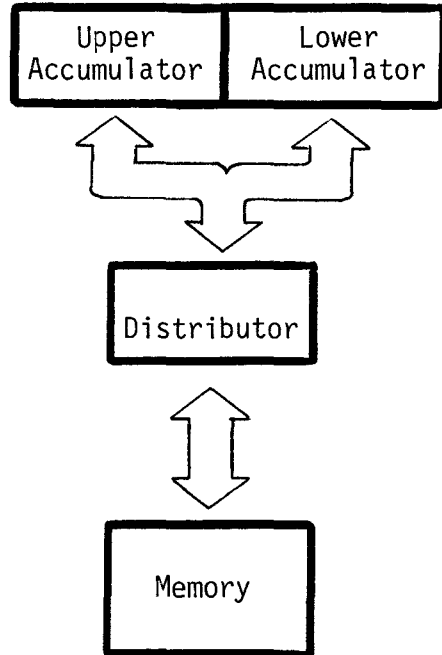
SYMBOLIC CODE FOR
CREATURE

00	TAIL	00	000	00	000	Image left when crawling.
01	SIZE	00	033	00	000	Length - 1.
02	INIT	RAL	K1	STD	MOVE	Initialize vector to not reproduce.
03		STU	TAIL	STD	TUM	Zero aft image and stomach.
04		STD	INDEX	RAL	K3	Initialize index for copy loop.
05		ALO	SIZE	STL	SENSE	Initialize vector to not feed.
06		RAL	K4	ALO	SIZE	Initialize vector to go back to SENSE after birth.
07		STL	BIRTH	TRA	SENSE	

08	K1	CRW	1	TRA	SENSE	Vectors for instruction at MOVE.
09	K2	CRW	1	TRA	REPRO	
10	TUM	00	000	00	000	Stomach counts food eaten.
11	INDEX	00	000	00	000	Index for copy loop.
12	K3	RAL	000	NZA	FEED	For construction of SENSE instruction.
13	K4	BIR	000	TRA	SENSE	For construction of BIRTH instruction.

14	SENSE	(RAL	033	NZA	FEED)	Go tro FEED if chunk of matter at nose.
15	MOVE	(CRW	1	TRA	SENSE)	Crawl 1 space. Go to SENSE or FEED.
16	FEED	RAL	TUM	ALO	ONE	Increment stomach.
17		STL	TUM	SLO	SIZE	
18		NZA	MOVE	STU	TUM	If stock not = size then go to MOVE.
19		LDD	K2	STD	MOVE	Otherwise, zero STOMACH, set vector to
20		TRA	MOVE	(Unused)		reproduce, and then go to MOVE.
21	REPRO	RAL	INDEX	ALO	ONE	Increment copy index and check for
22		STL	INDEX	SLO	SIZE	completion of copy.
23		NZA	COPY	STU	INDEX	Go to COPY if not done. Otherwise
24		LDD	K1	STD	MOVE	zero INDEX, reset vector at MOVE, zero
25		STL	TAIL	TRA	BIRTH	aft image and go to give birth.
26	BIRTH	(BIR	033	TRA	SENSE)	Bear, then re-enter main loop.
27	COPY	RAL	INDEX	ALO	K5	Set up copy instructions.
28		STL	MUTAT	TRA	MUTAT	
29	MUTAT	(LDD	xxx	MUT	TAIL)	Special mutation copy of a word to aft
30		TRA	SENSE	(Unused)		image.
31	ONE	00	001	00	000	
32	K5	LDD	000	MUT	000	For construction of MUTAT instruction.
33	NOSE	00	000	00	000	Forward sensing area.

SCHEMA OF RELEVANT 650 ARITHMETIC-LOGICAL UNIT ARCHITECTURE



OPERATION CODES USED IN INITIAL CREATURE

- RAL 65 XXX Reset entire accumulator to zero and load a copy of memory location XXX into the lower accumulator, leaving an image of the value in the distributor. Here as elsewhere, XXX is an address relative to the base address of the creature being executed.
- ALO 15 XXX Add the contents of memory location XXX to the lower accumulator, leaving a copy of the value from memory in the distributor.
- SLO 16 XXX Subtract the contents of memory location XXX from the lower accumulator. Distributor is left with a copy of the word from memory.
- STL 20 XXX Store a copy of the contents of the lower accumulator in memory location XXX. Distributor is left with a copy of the word stored.

STU 21 XXX The upper accumulator is stored in memory.
LDD 69 XXX The distributor is loaded with a copy of the word at memory location XXX.
STD 24 XXX The contents of the distributor are copied into memory location XXX.
NZA 45 XXX If the accumulator is not zero then control is transferred to the instruction at memory location XXX.

Non-650 instructions

TRA 00 XXX Control is transferred to the instruction at memory location XXX.
CRW 90 XXX Crawl XXX spaces.
BIR 99 XXX Notify Left Hand of God program of birth of a new creature with base location at XXX relative to base location of creature currently being executed.
MUT 54 XXX Store the lower accumulator at address XXX with a random mutation according to an input fixed rate.

annex A
description of the artificial universe
in Scientific American in 1985

description of artificial universe in *Scientific American* 1985¹

Inspired by a June 1959 *Scientific American* article on self-reproducing mechanisms by L. S. Penrose, Frederick G. Stahl of Chesterfield, Mo., created a miniature linear universe in which humble creatures lived, moved and (after a fashion) lived out their destinies. Stahl writes:
“Like Core War, I set aside a closed, linear segment of main memory in which a creature was simulated by modified machine language. The machine was an IBM Type 650 with drum memory. The creature was programmed to crawl through its universe eating food (nonzero words) and creating a duplicate of itself when enough food was accumulated. Like Core War, I had an executive program which kept track of who was alive and allocated execution time among the living creatures. I called it the “Left Hand of God.” Stahl goes on to discuss his program’s ability to reproduce. He also describes an interesting mutation mechanism; a program being copied might experience a small number of random changes in its code. However, Stahl reports, “I abandoned this line of work after one production run in which a sterile mutant ate and killed the only fertile creature in the universe. It was apparent that extraordinarily large memories and long computer runs would be needed to achieve any interesting results.”

¹ A.K. Dewdney, “Bestiary of viruses, worms and other threats to computer memories,” Computer Recreations, *Scientific American*, March 1985. The column with others was republished in A. K. Dewdney. *Armchair Universes: An Exploration of Computer Worlds*. (W.H. Freeman, 1987).

5 August 1984

Scientific American
415 Madison Avenue
New York, NY 10017
Attn: A. K. Dewdney

Dear Mr. Dewdney:

I found Core War as you discussed it in your May and August 1984 columns in Scientific American related to some work I did in 1960 at the Wayne State University Computing Center.

The attached paper describes that work. Except for some editing, the paper is essentially as written in 1961 including some rather dated references to computing machinery. I never sought to publish the work because I found little academic encouragement.

My work was inspired by an 1959 article in Scientific American wherein L. S. Penrose described John von Neumann's work on self-reproducing automata. I sought to create a digital analog to the physical entities studied by von Neumann. I also wanted to include aspects of evolution which was facilitated by the digital medium.

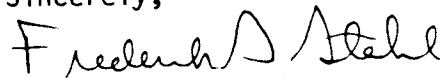
Like Core War, I set aside a closed, linear segment of main memory in which a creature was simulated by modified machine language. The machine was an IBM Type 650 with drum memory. The creature was programmed to crawl through its universe eating food (nonzero words) and creating a duplicate of itself when enough food was accumulated. Like Core War, I had an executive program which kept track of who was alive and allocated execution time among the living creatures. I called it the "Left Hand of God." A special operation informed the Left Hand of God of the birth of an offspring. The ability to reproduce is a significant difference between my little universe and Core War.

Another important feature is mutation. The interpreter in the Left Hand of God treated copy operations in a special fashion. For each digit copied, a pseudorandom number was computed and compared with a parameter I set. If the random number was less than the parameter, a random digit was written rather than the original digit. Thus the offspring creatures had random mutations in their programs at an average frequency determined by the input parameter.

I abandoned this line of work after one production run in which a sterile mutant ate and killed the only fertile creature in the universe. It was apparent that extraordinarily large memories and long computer runs would be needed to achieve any interesting results. Further, it seemed that concepts of energy, matter and work should be implemented to prohibit "free lunches." The work did raise some interesting theoretical and philosophical questions which are touched on in the attached paper.

I hope you will find them interesting too.

Sincerely,

A handwritten signature in cursive script that reads "Frederick G. Stahl". The signature is written in dark ink and is positioned above the typed name.

Frederick G. Stahl
14872 Sycamore Manor Drive
Chesterfield, MO 63017

annex B
digital computers at the Wayne
State University Computation
Laboratory in 1960

digital computers at the Wayne State University Computation Laboratory 1960

The Wayne State University Computation Laboratory was among the earliest computer research facilities in the United States with large-scale computers. The Lab was organized in 1949 under the auspices of the Industrial Mathematics Society (IMS) as a joint effort between the university and local industry. The Computation Laboratory performed three basic functions: 1) to provide service to industry and business, 2) to conduct academic research, 3) to provide training in computer operations.

Initially, the laboratory had a 1930-era Bush Differential Analyzer donated by the Massachusetts Institute of Technology. When it first began operation, the Lab focused on solving engineering problems for the automobile industry. A Unitized Digital Electronic Computer (UDEEC) was purchased from the Burroughs Corporation and put into operation in December of 1953. Increasing demand for computer services led to the purchase an IBM 650 computer in the spring of 1956.

UDEEC

The UDEEC was one of only a couple of machines special-built by the Burroughs Corporation. With 31 racks of electronics hidden behind large panels, the machine looked like a locker room. The electronic components behind each door were enormous by today's standards. Each flip-flop, for example, consisted of four big, hot vacuum tubes and their attendant resistors and capacitors, all mounted behind a roughly 6-inch-by-18-inch machined aluminum panel, complete with pilot lights to show its status—*on* or *off*. You would need an electron microscope to see the functionally equivalent component on a modern integrated computer chip.



IBM 650

The UDEEC was obsolete almost from the day it was delivered to the Lab. It had a mean operating time without an error of six hours. The workhorse machine at the Lab was the IBM 650, the “Magnetic Drum Data-Processing Machine Type 650.” It too was a vacuum machine but much more compact. The electronics were housed in three large boxes, each mounted on castors: console unit, power

supply, and card read-punch unit. Console unit, which also contained the CPU, was six feet high, five feet long, and three feet wide. It weighed 2,000 lbs. The power supply was in a similar box and weighed 3,000 lbs. The card reader and punch unit was smaller and lighter at 1,200 lbs. All units had removable side panels for maintenance. Contemporary photographs of the system seldom show big exhaust hoods over the console and power units to dissipate the heat generated by thousands of vacuum tubes.



The 650 was the first mass production computer. Two thousand were sold. At the time, I thought it was a relatively good machine to program. It was reliable and had a clean architecture. But it had its drawbacks. For example, debugging software involved laborious manual stepping through program instructions and observing the contents of CPU registers displayed on console lights. Input/output was another awkward process. The only input was via decks of IBM punch cards fed through the card reader/punch card unit. The machine had no connected printer. All output had to be first punched into cards and then printed on an IBM 402 tabulating machine, originally designed for standalone accounting applications.

The organization of the 650 random access memory is seldom seen today. Instead of addressable bytes, the memory was organized into 2,000 addressable *words*, each of which consisted of ten decimal digits and an arithmetic sign. As a benchmark, the 650's main memory would be equivalent to about 20 Kilobytes.

The IBM 650 was a "two-address" computer. Each machine-language instruction was organized into a two-digit operation code and two four-digit addresses. For example, the "STD" instruction

+24 1733 0324

would cause the CPU to store a copy of the ten digits and sign contained in the register called the "distributor" into word 1733 in the rotating drum memory. Then the CPU would take its next instruction from memory location 0324.

The machine could execute 44 different operation codes including: load a register from contents of a memory location, store contents of a register into a memory location, add, subtract, multiply, divide, shift, and branch. Typical execution times were 5 milliseconds per instruction.