Game of Life

About a year ago, I fascinated over a computer simulation called Conway's Game of Life. This simulation consisted of a two-dimensional grid. Its cells could be filled or empty, which corresponded to alive and dead, respectively. This grid followed four simple rules: any live cells with fewer than two live neighbors dies, any live cell with two or three live neighbors lives to the next generation, any live cell with more than three neighbors dies, and any dead cell with exactly three neighbors becomes a live cell. Filling in the grid, the player could create patterns with different properties. What I found fascinating was not the game itself, but the properties the game exhibited. One property was the game's demonstration of self-replication. If the live cells were placed correctly, then its cells could replicate without any intervention. In addition, the game could exhibit Turing-completeness, which is if a system of rules can simulate a single-tape Turing machine. In other words, a Turing-complete system has equivalence to a computer. This fascination led me to a crucial question that guided my research. Can a computer self-replicate?

I am not the only person to ask this question. John von Neumann asked this question in designing his universal constructor. John von Neumann's universal constructor consists of a self-replicating machine with twenty-nine states, which could perform data manipulation and logical operation. More recently, researchers in nanotechnology have been experimenting with molecular self-replicating structures. Applications of this technology could include medicine, chemistry, and electronics. For example, a molecular computer could search for cancerous cells and kill them, leading to a safer and more reliable cancer therapy. However, although research is optimistic, molecular self-

replication has proven to be quite difficult. Components can be expensive as well as the computer being disrupted by quantum effects.

This situation inspired me research alternative methods to nanotechnology. I decided to use cellular biology as a platform. Hypothetically, a cellular computer would share many applications of nanotechnology without its problems. A biological cell would reproduce predictably and perhaps carry its computational state with each generation. In addition, altering cells would be somewhat easier than a molecular computer. Using this as a basis, I began my research on a practical biological computer.

As I began my journey, I met with a molecular biologist, Dr. Katrina Brandis, about my interests. Collaborating with me, she recommended different journal articles to read. I found that many other researchers had tackled the same problem, although most of their research was theoretical. One researcher, for example, had designed a theoretical biological computer. The computer's function was similar to a Turing machine. Inspired by this device, I decide to implement the design in an E. coli culture. Unfortunately, I learned quickly that, although the device was viable in theory, its implementation was impractical. While the device had all the characteristics of a working computer, it was slow and difficult to scale. My progress had come to a halt.

Without any problems to solve, I saw this as an opportunity to increase my background in both computer science and biology. Reading an article a day, I took notes on every new concept. Some articles were useful; however, most were irrelevant. With every few articles, I discovered different computer architectures, computational models, and proteins. Along with my notes, I continued to collaborate with Dr. Brandis. After several

weeks, I was able to divide my research into subproblems. These categories could be describe as the arithmetic logic unit, the control unit, the data memory, and the instruction memory. These categories were based on the Harvard architecture. The Harvard architecture, which has little relationship with Harvard University, is the computer architecture that powered the Harvard Mark I punch-card computer. Its operation could be divided into four sections: the arithmetic logic unit, control unit, data memory, and instruction memory. Much like Conway's Game of Life, this architecture could perform any operation of a desktop computer. Thus, if I could design a biological computer with a Harvard architecture, then it could perform any operation of a desktop computer.

Finally, I had a distinct set of goals to approach. Beginning with arithmetic logic unit, I designed each section of the computer. As I did so, I made sure each component would work independent of the others. In this way, I could avoid focus on each section without needing knowledge of other parts. Using my notes, I read through every hypothesized biological part, removing each dysfunctional part. Because I needed efficiency in space and time, I optimize these biological parts and created the arithmetic logic unit. Next, I had to find a way to read and store memory. The difficulty was enumerating each register, or memory block. However, since recursion is nearly equivalent to iteration, I implemented this concept into the data memory. Lastly, I had implement the computer's programs in a precise manner. Considering concepts from the arithmetic logic unit and data memory, I design the instruction memory and the control unit to work together.

While I had solve my set of subproblems, I still had to connect each of the sections. Luckily, since the sections were independently isolated, I didn't have to worry about

conflicts between them. Using biological signals, I was able to communicate between each of the sections. Finally, I had completed a viable biological computer. Going through each process, I made sure everything would work. Giving myself a pat on the back, I knew I was ready for experimentation.

Immediately, I knew I could not implement the entire computer. I did not have enough time to create every part from scratch, but I still wanted to test its validity. While I could not test the computer in its entirety, I could still test its individual parts, for the computer could be divided into only several parts. Thus, if I successfully created each of the parts, I could hypothetically create the entire computer. At the O'Hara laboratory at my school, I tested each part rigorously to ensure its validity. I was please when everything worked as it was designed. As I tested the final part, I knew my research had come to an end.

The Biological Computer

Since Charles Bennet likened the operation of the RNA polymerase to a Turing machine in 1973, biological computation has been an ever growing field. Since then, both Ehud Shapiro and Paul Wilhelm Karl Rothemund have independently designed biological Turing machines. While a biological Turing machine shows great promise in biological computing, it has a few weaknesses. First, a Turing machine, not necessarily biological, is very difficult to program. This is because the Turing machine uses states to compute on a tape unlike microinstructions on a central processing unit. A set of instructions would have to be reduced into a set of states that the Turing machine could read. In addition, a Turing machine is extremely slow. Unlike a central processing unit, a Turing machine reads data on a tape and therefore computes only as quickly as it moves on the tape. On the other hand, a central processing unit uses registers with addresses allowing it to compute as quickly as a clock cycle.

In the biological computer, the most fundamental component of the biological computer is the serine integrase. If engineered appropriately, the serine integrase can function as a transistor. In conjunction with the Harvard architecture, these serine integrases allow the biological computer to be Turing-complete, or computable. The Harvard architecture consists of four sections: the arithmetic logic unit, control unit, data memory, and instruction memory. The arithmetic logic unit is the “brain” of the computer; it takes data as well as instructions to output results, whether they are basic mathematical calculations or intensive image rendering. In each operation, the data is retrieved from the

data memory. In this way, data is constantly recycled between the data memory and arithmetic logic unit. To direct this cycle, the control unit retrieves microinstructions from the instruction memory. The control unit communicates with the arithmetic logic unit and data memory to implement these instructions. Communication between each section allows the Harvard architecture to compute complex operations.

To design the biological arithmetic logic unit, the architecture implements Boolean logic gates. These components perform logical operations such as inclusive-or, exclusive-or, and not. Coupling the logic gates, the arithmetic logic unit can perform mathematical and logical operation, which are both crucial to Turing-completeness. The data memory also uses Boolean logic gates. Each register, or block of memory, consists of a master section and a slave section. Using Boolean logic gates, the master section stores data while the slave section reads it. To direct the process, the control unit retrieves microinstructions from the instruction memory, which has similar memory operations to the data memory. These instructions communicate with the data memory to distinguish where to store and retrieve data. In addition, these instructions also direct which arithmetic logic operations to use on data. These processes encompass the Harvard architecture to create a biological computer.

This biological computer is made possible by serine integrases. Using serine integrases, it is possible to create a biological transistor. These biological transistors are the basic components in a Boolean logic gate. Designed along the Harvard architecture, these Boolean logic gates create the arithmetic logic unit, control unit, data memory, and instruction memory. With the sections communicating together, the biological computer

can be Turing-complete. This biological computer offers many applications to both chemistry and medicine. For example, using human-engineered proteins, it is possible to synthesize various organic chemicals. With biological computing, new avenues in science can be opened.