TJHSST Computer Systems Lab Senior Research Project
The Engineering of Functional Designs in the Game of Life
2007-2008

Liban Mohamed

June 6, 2008

Abstract

Conway's Game of Life is a set of rules in a two dimensional cellular automata grid. This ruleset was specifically chosen by John Conway for the ability to create stable patterns as well as the difficulty of creating patterns which grow without bound. This difficulty was rather quickly overcome by Bill Gosper's glider gun, which opened up the ability to create binary computational devices such as logic gates. As soon as the possibility of binary computational devices in the Game of Life was discovered, it was realized that patterns could be designed in the Game of Life which could symbolically carry out computations. This project endeavors to facilitate in the design and creation of functional patterns in the Game of Life.

1 Introduction

1.1 Scope of Study

This project involves the design and coding of a high functioning cellular automata interface. The purpose of the creation of this interface is the assistance it is able to provide with the design and creation of functional patterns in cellular automata. Ideally, this interface would endow a user with the capability to design a pattern to do anything in any given cellular automata ruleset that is possible in said ruleset as well as display the evolution of that pattern. This interface is broken down into two parts which help with two goals of the user: the design, and the creation of patterns. The creation of patterns in the game of life is assisted by a highly flexible cellular automata editor, with features so abundant they're falling out of its ears. The second portion of the interface is a program which assists in the design of functional patterns by providing the user with a way to search for patterns that evolve into user-specified patterns in a given number of generations.

1.2 Type of research

This project involves pure applied research. None of the problems that this project attempts to address have not been tackled before and the trick is to find, using computer searches, patterns which have the functionality that will be required. This will all be new to me, but it is the nature of cellular automata that fundamental understanding is impossible to achieve: cellular automata are notable precisely for the ability to defy prediction.

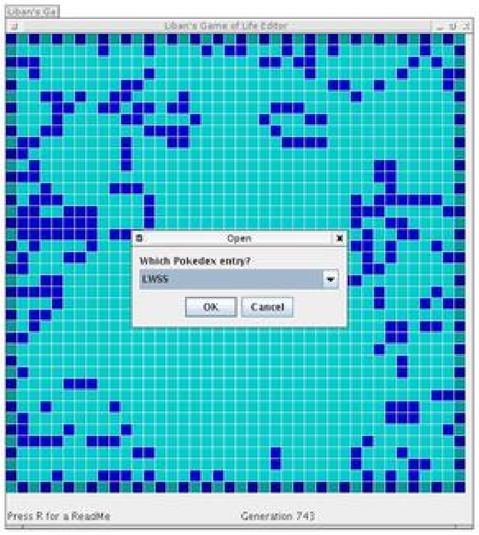
2 Background

As previously mentioned, a man by the name of Paul Rendell spent a number of years developing a finite Turing machine extensible to universality in Conway's Game of Life. His project is the extreme of complexity, but others have created logic gates, most notably Andrew Adamatzky's LogiCell, presented in his *Collision Based Computing*. With respect to Rendell, two rather large extensions are possible on his work. Firstly, he created his Turing machine with extensibility to universality specifically in mind, meaning one could use his Turing machine as a template to create a universal Turing machine. In addition to that, Rendell is currently working on create a stack cell generator, which would generate tape for his Turing machine, effectively giving it infinite tape.

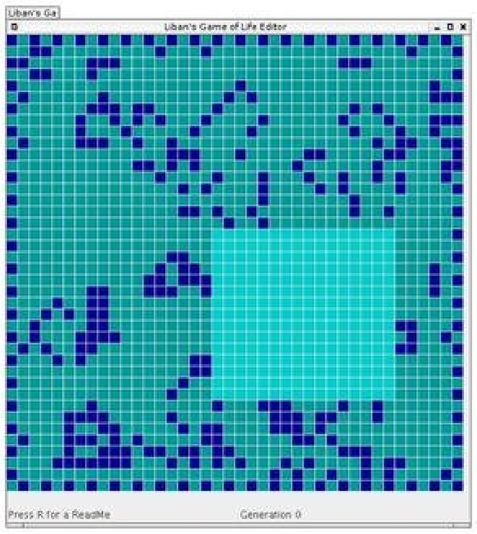
This project differs from the other projects in that it endeavors to create not an end product such as a Turing machine or specific logic gates but rather a method of producing these end products and subsequently showing them off to friends. The end product of this project is capable of reproducing others' results as well as generating some of its own.

3 Development

The current edition of my cellular automata editor and runner functions to run any 2 dimensional cellular automata setup using von Neumann neighborhoods. The program is highly efficient, running at up to 200Hz in the Systems Lab computers. The inputs may be either clicked into the grid or loaded from a text file of on and off states. In the interface, a number of useful functions have been implemented, including the selection and running of sub-grids, copying and pasting rectangular areas of the grid, clearing selected areas, and the aforementioned saving and loading of patterns. Screenshots displaying the loading of a pattern and the clearing of a selected area are displayed here.



(a) Loading a pattern



(b) Clearing an area

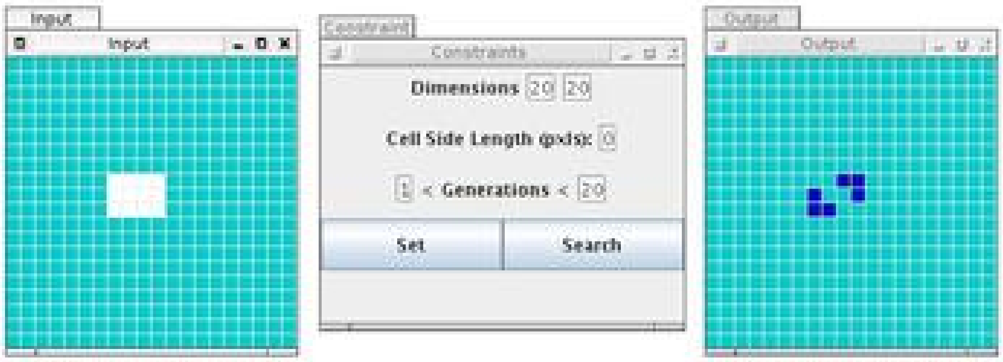
The other part of the cellular automata interface, the search function, is functional as well. This part of the interface presents the user with two grids of cells, which the user then fills with different sorts of cells - alive, dead, unspecified, or self-referential - and then asks to find all pertinent inputs. For the input grid, cells that are alive are constrained to be alive in any of the possible inputs that the program supplies, cells that are dead are constrained to be dead in any of the possible inputs that the program supplies, and cells that are unspecified can be either alive or dead in any of the possible inputs and thus require the program to check both scenarios. For the output grid, cells that are specified to be dead must be dead at the end of the run of each possible input, cells that are specified to be alive must be alive at the end of the run of each possible input, and it does not matter whether the cells that are unspecified are alive or dead. For the self-referential cells, which may only be placed in the output grid, the state of that cell at the end of the run of each possible input must be the same as it was at the time of the input. As this idea is rather cumbersome in words, images of various searches are included in the results section. It should be noted that this portion of the program also implements the various functions that the editor implements, such as copying and pasting, with the exception of writing patterns to and loading patterns from a file.

4 Results and Discussion

Very little attention will be given to the portion of the program which runs cellular automata patterns as either the function that it serves needs no explanation or the reader has no hope of understanding any part of anything else that will be said and should devote themselves to more fruitful endeavors than reading this paper.

For the search program, there is a good deal of discussion to do. So that the user fully understands the functioning of the search program, I will go into some detail about how it is used here. The most trivial use of the search function is the verification that one arrangement of dead and alive cells leads to another arrangement of dead and alive cells. This setup involves no unspecified cells, and the user receives a quick yes or no answer as to whether the pattern specified in the input becomes the pattern specified in the output at any point in the time frame delineated by the user.

The next use of the search program would be if a user knew the shape of a pattern that evolved in a particular manner, but did not remember the state of a smattering of the cells. If this were the case, the user would input as much of the input as he knows, placing unspecified cells where knowledge is lacking, and fill in the output grid completely, such as in the following screenshot.

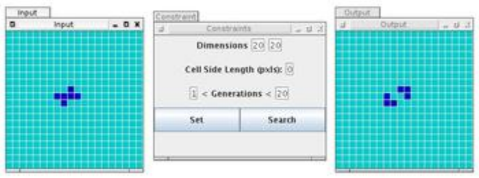


Dimensions 20 20

Cell Side Length (pxls): 0

1 < Generations < 20

This particular search produces two results, both of which are supplied by the program and are displayed here. If the user had only cared about a

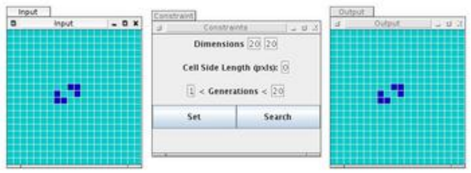


Dimensions 20 20

Cell Side Length (pxls): 0

1 < Generations < 20

(a) The first result



Dimensions 20 20

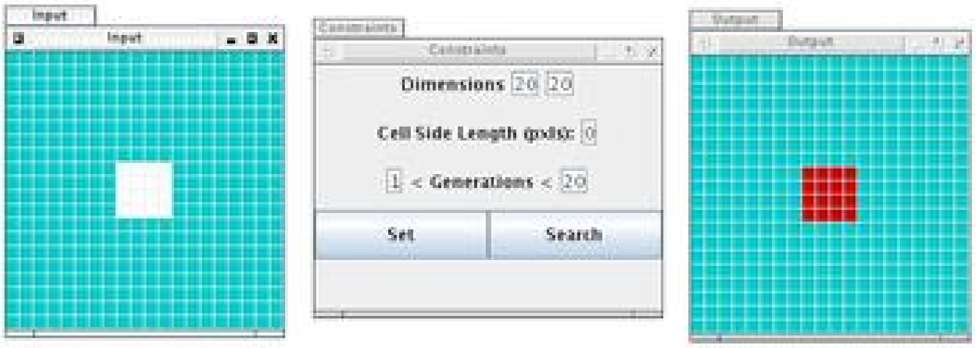
Cell Side Length (pxls): 0

1 < Generations < 20

(b) The other result - look familiar?

certain part of the output, the user could make the rest of the cells in the output unspecified.

The self-referential cells were designed with one thing in mind: oscillators. In order for an output to match an input, the state of these cells must be the same as it was in the input. To search for every oscillator or still life in a four by four square, the user would make a four by four grid of cells in the input unspecified, and then make the corresponding square of cells in the output self-



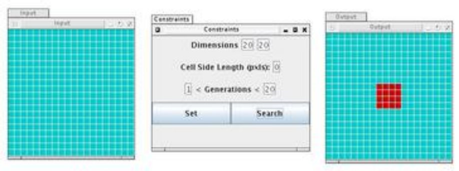
Dimensions 20 20

Cell Side Length (pxls): 0

1 < Generations < 20

referential.

This search returns 123 results, which may seem surprising given the simplicity of the result. This occurrence can be explained by the fact that every result appears multiple times rotated and translated about the area of unspecified cells, giving many redundant results.

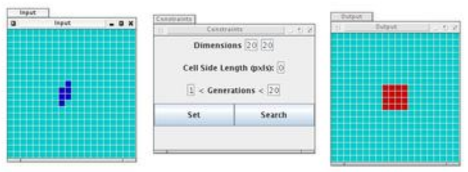


Dimensions 20 20

Cell Side Length (pxls): 0

1 < Generations < 20

(a) Result 1

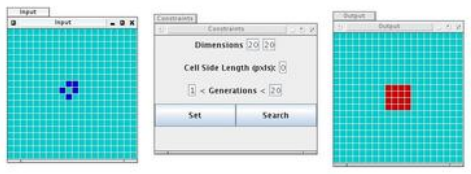


Dimensions 20 20

Cell Side Length (pxls): 0

1 < Generations < 20

(b) Result 15

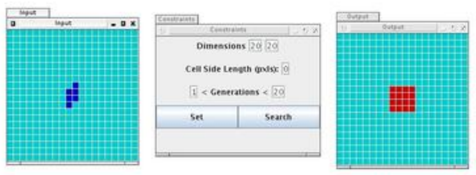


Dimensions 20 20

Cell Side Length (pxls): 0

1 < Generations < 20

(c) Result 23



Dimensions 20 20

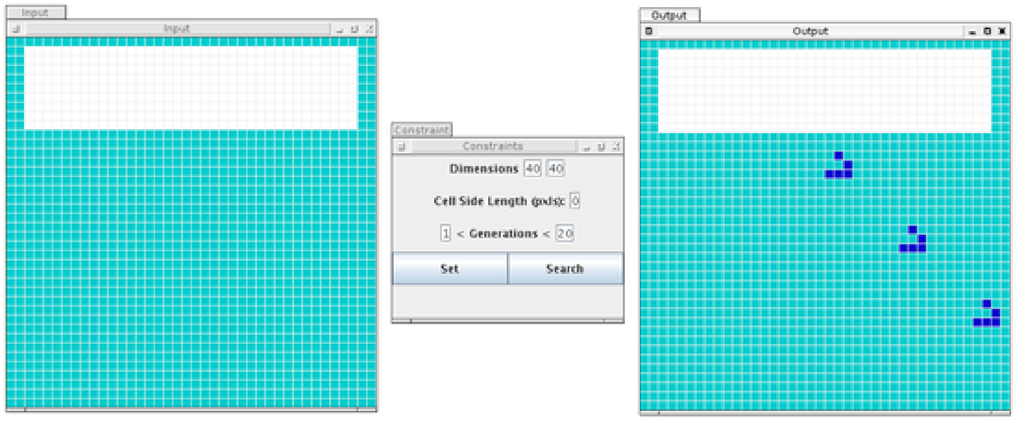
Cell Side Length (pxls): 0

1 < Generations < 20

(d) Result 57

This first result is an area of dead cells, all of which do indeed return to their initial state. The fifteenth result is an oscillator with a period of two generations. The twenty-third result is another still life. The fifty-seventh result is a translation and rotation of the fifteenth result.

The last sample search that I have designed would actually be useful, and is designed to verify the existence of the Gosper gun. To produce this result, the user places a square of unspecified cells exactly the size of the Gosper gun in one area of the input, and then manually draws in a stream of gliders exhibiting the same period as is found with any Gosper gun. In the output, the user fills the area intended to be the Gosper gun with self-referential cells, and makes the same stream of gliders, but makes it a few gliders longer.



Dimensions 40 40

Cell Side Length (pxls): 0

1 < Generations < 20

Notable for users attempting to perform operations akin to the one detailed in the last example, the time a given search takes is an unfortunate function of the size of the grid, the maximum number of generations allowed, and the number of unspecified input cells. The last variable is the most important in the function for any number of cells more than a handful, and the dependence of time on the number of unspecified cells is a big O of 2^n, as each new unspecified cell multiplies the number of possible inputs by two. Fifteen unspecified cells comes with a reasonable run-time; the program takes around half a minute to run. Twenty unspecified cells takes some fifteen minutes. Running a search program with four hundred unspecified cells, around the number of cells needed to contain a Gosper gun, requires more than a googol years.

Another consideration for users of the search program is the number of possible inputs that function to go from an input to a fairly generic output. For example, the search for a blank grid detailed earlier presents the user with 20,037 possibilities. Though most outputs are more specific than an entirely blank grid, no user I know wants to sift through tens of thousands of arrangements of cells to get to the one he wants. At the moment, all that the user has to combat the constraints on the number of unspecified cells used and the number of possible inputs generated are his own wits. It is absolutely necessary for the user to be familiar with the uses of the various constraints so that the solution set is as limited as possible. The setup we examined to search for the Gosper gun, for example, was rather clever. Though I didn't have the googol years to test it, I'm fairly certain that the search would have return the Gosper gun and only the Gosper gun.

4.1 Conclusion

The purpose of this project was to create a program which would assist in the design and creation of cellular automata patterns. In some ways, especially for relatively trivial sizes of patterns, this project was immensely successful. If the user can find a way to determine the state of all but 15 cells in a desired pattern, use of the program will be swift and successful. The capability of the project to supply in one go large constructs such as hershel tracks and glider guns, however, is limited by the nature of cellular automata and the nature of the hunt.

If the success of this project must be reported on in a quantized manner where the only acceptable answers are success or failure, this project would be rated a success as it demonstrated much of the methodology used in the type of work that it is meant to perform. That is not to say that there are no possible extensions, though...

4.2 Recommendations

The recommendations that I have for this project for the most part involve extensions and modifications to the second part of the project, the search function. Firstly, I would translate the code to C, and modify the data structures so each cell is represented by a bit. This change would have the advantages of restricting the amount of memory required to run the program and also decreasing the amount of time required due to the speed of bit operations.

The second set of recommendations that I have for anyone interested in expanding this project would be to do some work in pattern recognition. Pattern recognition could benefit the functioning of this project in two ways. Firstly, if done correctly, pattern recognition can be used to increase the speed with which the program is able to evolve the grid of cells through generations. Secondly, pattern recognition could be used to eliminate a large number of extraneous possibilities in searches which involve large areas of contiguous unspecified cells. This is the real gem of pattern recognition, and could potentially be used to reduce the big O from 2^n to something significantly more manageable for large n.

The last set of recommendations that I have for the extension of this project are more encouragement than than anything. At the commencement of this project, my goal was to create a program which would go as far along providing every capability that a user could wish for in designing patterns for cellular automata. I have not completed this goal, and I do not believe it is possible to complete it without infinite amounts of time for coding and infinite computational capability. However, there are a number of small next steps which I was not able to take but could be easily implemented by anyone with the gumption. One would be some elegant way to deal with the edges in the search function. Another would be an elegant combination of the two separate programs that I have created for the task. One could also allow the user to check the grid at multiple times using or, and, and xor to connect these different arrangements at different times. The most innovation remains to be done in the creation of different types of cells. In the search function, I have defined two new types of cells: unspecified and self-referential cells. There are many, many more possibilities, though, such as cells which refer to *other* cells, either in their input or output stages, in checking for the verification of a possible input.

References

[1] www.rennard.org/alife/english/logicellgb.html

[2] Collision Based Computing, by Andrew Adamatzky