The Engineering of Functional Designs in the Game of Life

Computer Systems Lab 2007-2008

June 10, 2008

Liban Mohamed

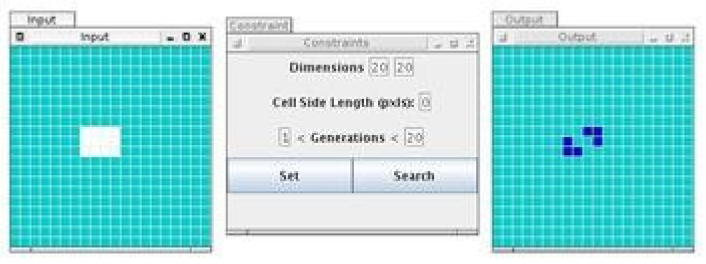
Abstract

First, this project endeavours to create a flexible and powerful Game of Life interface. After that is achieved, this project goes on the create search programs for patterns in the Game of Life. Finally, this project intends to use the functionality enabled by the previous two steps to design a pattern which can be used for computation in the Game of Life. That is, the purpose of this project is to create one or more patterns in the game of life which take an input in the game of life and consistently produce an output which can be interpreted to get the right answer.

Results

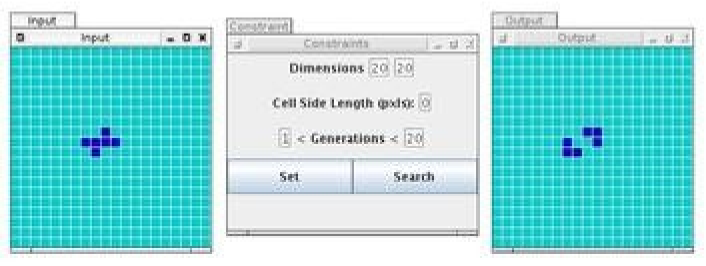
For the search program, there is a good deal of discussion to do. So that the user fully understands the functioning of the search program, some depth will be gone into here. The most trivial use of the search function is the verification that one arrangement of dead and alive cells leads to another arrangement of dead and alive cells. This set-up 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 few of the cells. If this were the case, the user would put in 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 one of which is displayed here.



Dimensions 20 20
Cell Side Length (pxls): 0
1 < Generations < 20

Another sample search that has been designed, one which is actually quite useful, 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.

Notable for users attempting to perform operations akin to the one detailed in the last example, the time a given search takes is a 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 dominant variable in the function for any number of unspecified cells greater than half a dozen, 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 runtime; 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 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, it is unreasonable to expect a user to sift through tens of thousands of results to find the desired arrangement. 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.

Introduction

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 endeavours to facilitate in the design and creation of functional patterns in the Game of Life.

Conclusion

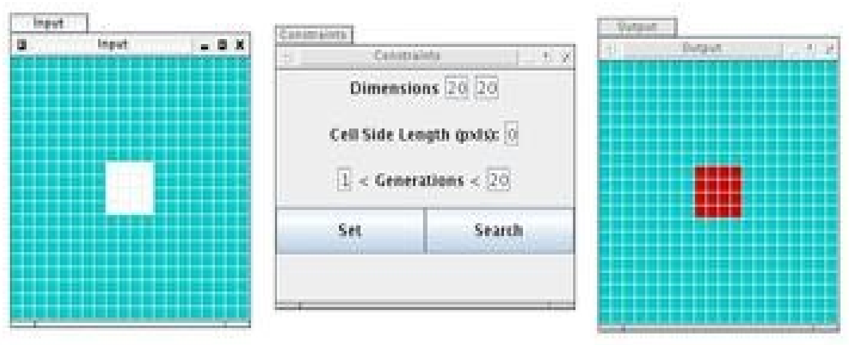
The purpose of this project was to create a program which would assist in the design and creation of cellular automata patterns. In some ways, especially for relatively trivial sizes of patterns, this project was successful. If the user can find a way to determine the state of all but 15 cells in a desired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

Recommendations

For the most part, the recommendations that I have for this project 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 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.



Dimensions 20 20
Cell Side Length (pxls): 0
1 < Generations < 20