SYSC 5104 METHODOLOGIES FOR DISCRETE EVENT MODELLING AND SIMULATION

Assignment #2

Part #2

CELL-DEVS MODELLING OF RULE 150

Student # 100868021

Colin Timmons

15 November 2011

TABLE OF CONTENTS

TABLE OF FIGURES

TABLES

1. Part I

1.1 Identification

The cellular automata that will be modelled for this assignment is gathered from Stephan Wolfram's website (accessed 30 Oct 11).[1] In particular, the model to be implemented into Cell-Devs is Boolean model - Rule 110. This modelling of information is more designed towards developing a thorough understanding of the modelling tool and its aspects rather than the complexity of its model.

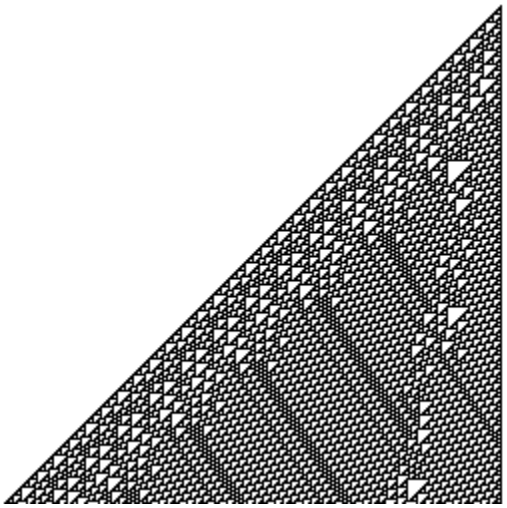


Figure 1. Boolean Model Rule 110[2]

Current cell	111	110	101	100	011	010	001	000
New state of centre cell	0	1	1	0	1	1	1	0

In the future, after the assignment delivery, I would like to build on the amount of effort required and recoding necessary for the tool to construct increasing multiple dimensions with this simple rule as an example.

[1] http://www.stephenwolfram.com/publications/recent/architecture/
[2] http://en.wikipedia.org/wiki/File:CA_rule110s.png

2. Part II

2.1 Formal Specification

The formal specification <X, Y, S, N, type, d, τ, δint, δext, λ, ta> for the Cell-DEVS Rule 110 model is defined as follows:

X = {∅}

Y = {∅}

S = { [0, 1] }

N = { (0,0), (-1,-1), (-1,0), (-1,1) }

Type = transport

d = 10ms

δint, δext, λ and ta are defined using Cell-DEVS specifications

2.2 Neighbourhood List

(-1,0)	(0,-1)	(1,1)
	(0,0)	

Table 1. Neighbourhood List

As Table 1 demonstrates, the neighbourhood of the cell is the immediate upper left, upper centre and upper right neighbours. The value of the contents of these cells determines the value of the desired cell.

Rule 110 is called 110 because the active or '1' condition of the neighbourhood cells can be related to their combined binary value as demonstrated in Table 2.

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

Table 2. Rule 110 Binary Logic

The possible output states in binary 01101110 = 0*2^7 +1*2^6 + 1*2^5 +0*2^4 +1*2^3 +1*2^2 +1*2^1 +0*2^0 = 110. Recursively, Rule 110 is an elementary cellular automata and each row develops from the previous row. Continuing this process, the developed automaton generates into a class 4 behaviour that is neither stable nor chaotic.

2.3 Detailed Description

Rule 110 has been described as capable of universal computation. At its simplest, Rule 110 is a simple Turing machine, capable of universality which means that many of their properties will be non-decidable, and not amenable to closed-form mathematical solutions.

The function of the universal machine in Rule 110 requires an infinite number of localized patterns to be embedded within an infinitely repeating background pattern. The background pattern is fourteen cells wide and repeats itself exactly every seven iterations. The pattern is 00010011011111.

Three localized patterns are of particular importance in the Rule 110 universal machine. They are shown in the image below, surrounded by the repeating background pattern. The leftmost structure shifts to the right two cells and repeats every three generations. It comprises the sequence 0001110111 surrounded by the background pattern given above, as well as two different evolutions of this sequence.[3]

As shown in Figure 2, these patterns are present and highlighted in orange and red respectively.

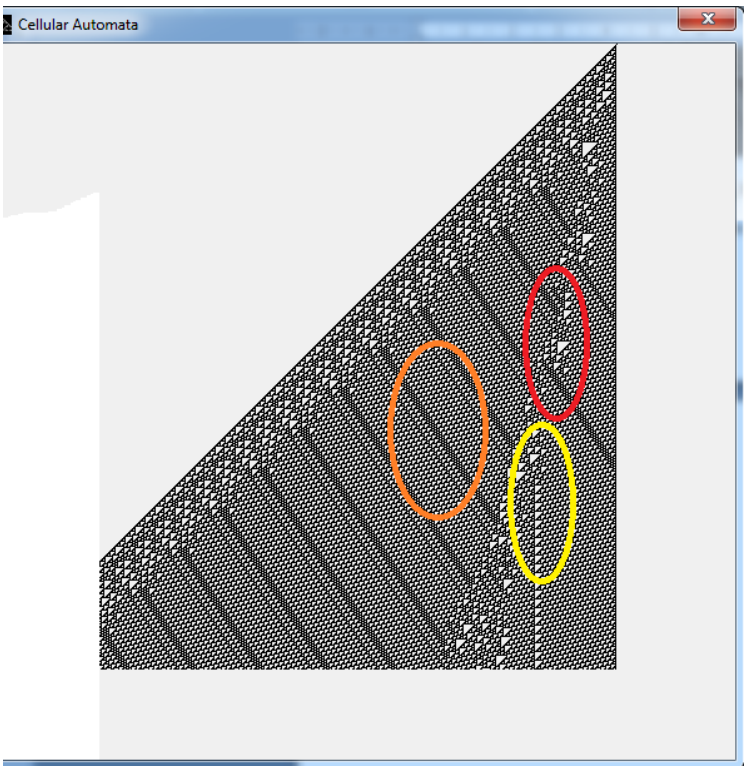


Figure 2. 600 by 600 Rule 110 Model

The yellow circle represents the interaction of these two structures as they pass through each other without interference to produce a third unique structure, deemed as self-reproduction.

[3] http://en.wikipedia.org/wiki/Rule_110

2.4 Implementation

Since Rule 110 is a binary combination of the previous neighbour's data value, the binary sequence of the neighbour's data value were placed into a Karnaugh Map to reduce the equation into its simplest representation as a product of sums.

Thus, letting A represent the upper left neighbour

B represents the upper centre neighbour

C represents the upper right neighbour

Rule 110 logic was replaced from:

Rule 110 = ABC + AB|C + A|BC + A|B|C + |ABC + |AB|C + |A|BC + |A|B|C

to

Rule 110 = |AB + |BC + B|C.

In terms of CD++, this rule was rewritten as

```
rule : 1 10 { ( ( (-1,0) = 0 and (-1,1) = 1 ) or ( (-1,-1) = 0 and (-1,0) = 1 ) or ( (-1,0) = 1 and (-1,1) = 0 ) ) }
```

Rule 110's state is either a 1 or a 0. The opening row has the initial condition that all data values are 0 except for the rightmost digit which is a 1. This implementation was the simplest to clearly demonstrate patterns listed in Section 2.3. With this input, the data evolves into the pattern shown in Figure 2.

In order to perpetuate Figure 2, a transition rule was implemented that the cells retain their changed state from a non-determined state (i.e. 0 or non-calculated) to a 1. This rule was

```
rule : { (0,0) } 100 { t }
```

2.5 Cell-Devs Coupled Model

Thus the coupled model description for the Rule 110 was as follows. Note the comments on dimensions as the model is set up for 100 iterations:.

```
[top]
components : Rule110

[Rule110]
type : cell
%Change dimensions for different size of iterations
dim : (100,100)
delay : transport
defaultDelayTime : 10
border : nowrapped
neighbors : Rule110(-1,-1) Rule110(-1,0) Rule110(-1,1) Rule110(0,0)

initialValue : 0
%change Rule110.val for different dimensions as spcified above
initialCellsValue : Rule110.val
localtransition : Rule110-rule

[Rule110-rule]

% Karnaugh mapping the boolean equation reduces it from
%   |A|B|C + |A|B C + |A B|C + |A B C +  A|B|C +  A|B C +  A B|C +  A
B C
% where A = (-1,-1)
% where B = (-1,0)
% where C = (-1,1)
% to
%   |BC + |AB + B|C
rule : 1 10{ ( ( (-1,0) = 0 and (-1,1) = 1 ) or ( (-1,-1) = 0 and (-
1,0) = 1 ) or ( (-1,0) = 1 and (-1,1) = 0 ) ) }
rule : { (0,0) } 10 { t }
```

2.6 Implementation Problems

CD++ on the eclipse platform, was able to produce 200 iterations of Rule 110 fairly fast. At this level, the structures listed in section 2.3 were not fully visible; however, the gliders or spaceships pattern was developing. Raising the iteration above 400 took the CD++ simulator 6 to 8 hours on a I7-2600K over clocked to 4.2 GHz with 16 Gig of memory. The transition was lowered to 0.01 ms in order to attempt to speed up the processing time; however, though beneficial to the execution time, the Rule110DRW_3_log produced an immediate transition time causing the drawlog executable to produce only the final state output rather than the transitional growth diagrams.

The eclipse platform installed with CD++ Builder is a virtual 32 bit machine that has a limited default java heap space of 40 Megs of memory. As this assignment required a default configuration, the testing time was not conducive to multiple test cases of high resolution. Raising the iterations further caused, out of memory problems with eclipse and termination of the platform. Specially, the size formatting of the drw files were unable to be rendered using the text editor in java. Microsoft wordpad was used instead because of this limitation.

As the Boolean logic of Rule 110 and its accompanying neighbours was logically easy to program, a Boolean rule generator was coded that simplified and produced the same output using the pixels of a dialog to represent the states changes for low and high end resolution and is discussed next.

3. Boolean Rule Generator

A Boolean logic generator was created using Visual Studio 2010 to verify that the rules in CD++ were correct and that the system would develop correctly to overcome the speed limitation of the eclipse platform. The data between the two systems, i.e the drw output to the log output representation were compared. In the random samples checked, the two outputs were examined and verified as correct and identical.

The Boolean Rule Generator can produce any of the 256 possible combinations besides Rule 110 and is submitted as a development tool for cellular automata.

The setting of the pixels was used as compared to z-buffering a bitmap to show the developmental growth of the cellular automata rather than the final output. The program is designed to increase the dialog size to match the size of the rows and columns.

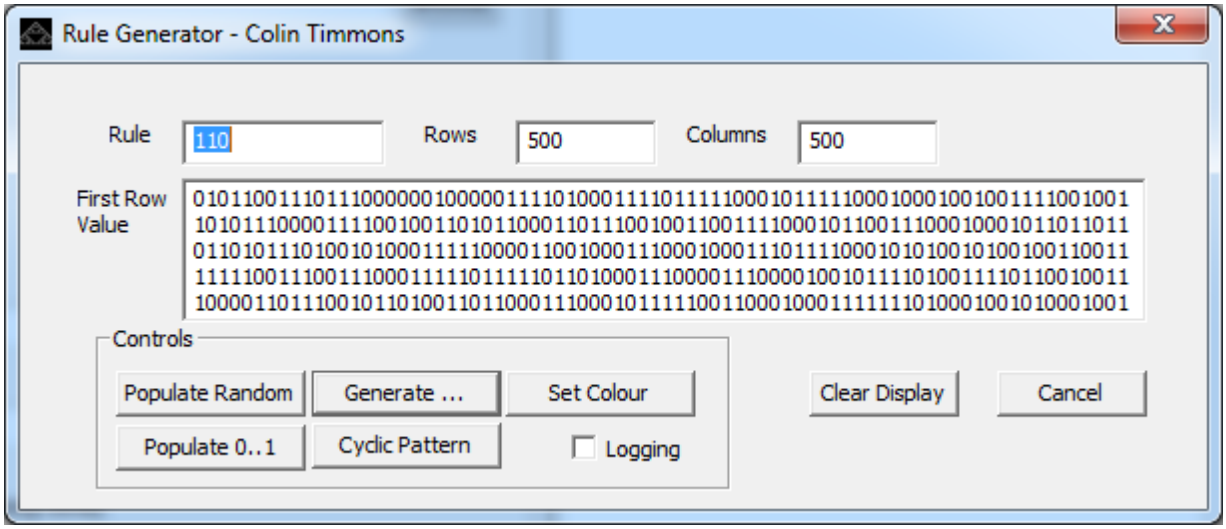


Figure 3. Boolean Rule Generator

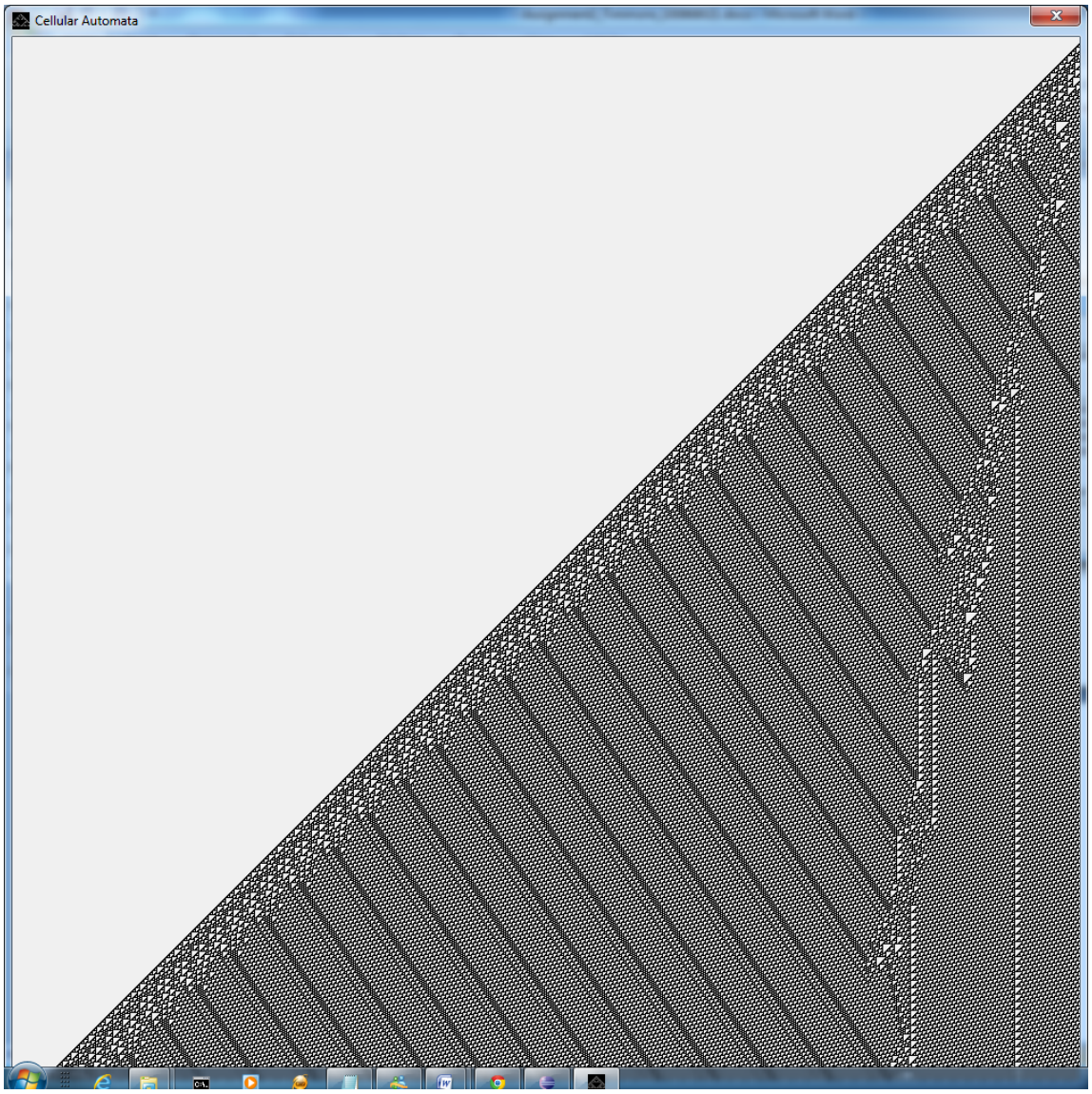


Figure 4. Boolen Rule Generator Output

Comparison example at line 28, using cd++ produced the first row output while the Boolean Rule generator logger produced the second row.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	1	0
0	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	1	0

Table 3. Row Comparison between CD++ Modeller and the Boolean Rule Generator

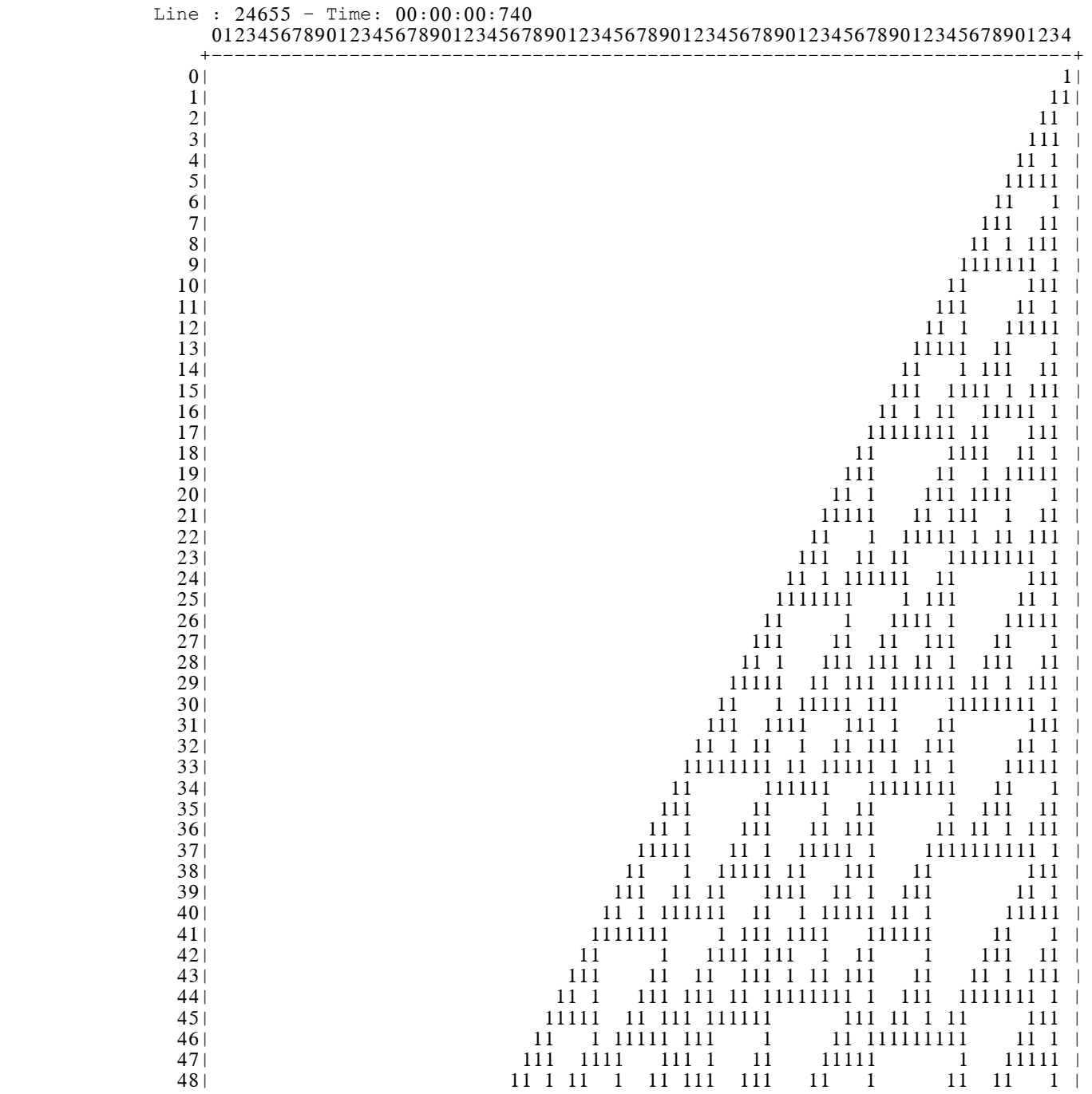
The neighbours in the Boolean Rule Generator are set for the previous neighbours. as shown in the snippet of code to determine the rule and the neighbours' values.

```
////////////////////////////////////////////
//     Method: DetermineRule
//     Attributes: int - position of the cell in the vector
//     Function: Using the rule entered on the dialog
//                     the method compares the loading of the
//                     previous row cells with the present cell
//                     in question and returns true for
//                     population of the vector if the
//                     conditions are correct
//     Return: bool - true if successful
////////////////////////////////////////////////
bool CRuleDlg::DetermineRule( int nValue )
{
	int nBinary = 0;
	int mask = 1;
	char nTempChar[3] = {0,0,0};
	char nTempNeighbour[3] = {0,0,0};

	//The value of each column for the row was passed through from the vector
	//Assign stack memory to the heap memory value for speed
	nTempNeighbour[2] = vecRow.at( nValue - 1);
	nTempNeighbour[1] = vecRow.at( nValue);
	nTempNeighbour[0] = vecRow.at( nValue + 1);

	//Range checked at the endfocus method but safety first
	if( nRule <= 0)
		return false;

	//check the value of each bit
	for( int i = 0; i < 8; i++)
	{
		//determine if a one is present in the rule
		nBinary = nRule & mask;

		if( nBinary )
		{
			//depending on the position the neighbour will have a specific value
			m_Neighbour[2] = ( i >= 4) ? 1 : 0;
			m_Neighbour[1] = ( i & 0x2 ) ? 1 : 0;
			m_Neighbour[0] = (i & 0x1 ) ? 1 : 0;

			//Now compare to actual data through memcmp to see if the value
matches the theory
			char nCompare = memcmp( nTempNeighbour, m_Neighbour, sizeof(
nTempNeighbour ) );

			//memcmp returns 0 if true so return success
			if( 0 == nCompare )
				return true;
		}
		//Set the mask higher to get the next bit
		mask <<= 1;
	}
	//no valid comparison so return false
	return false;
}
```

4. Testing

Samples of various dimensions were examined. Concise generation of output structure and their transition and development was best developed by a single setting of a one value input on the top row.

For example, the final output of the drw file for a 75 by 75 cellular automata is as follows. The formatting has been modified for readability.



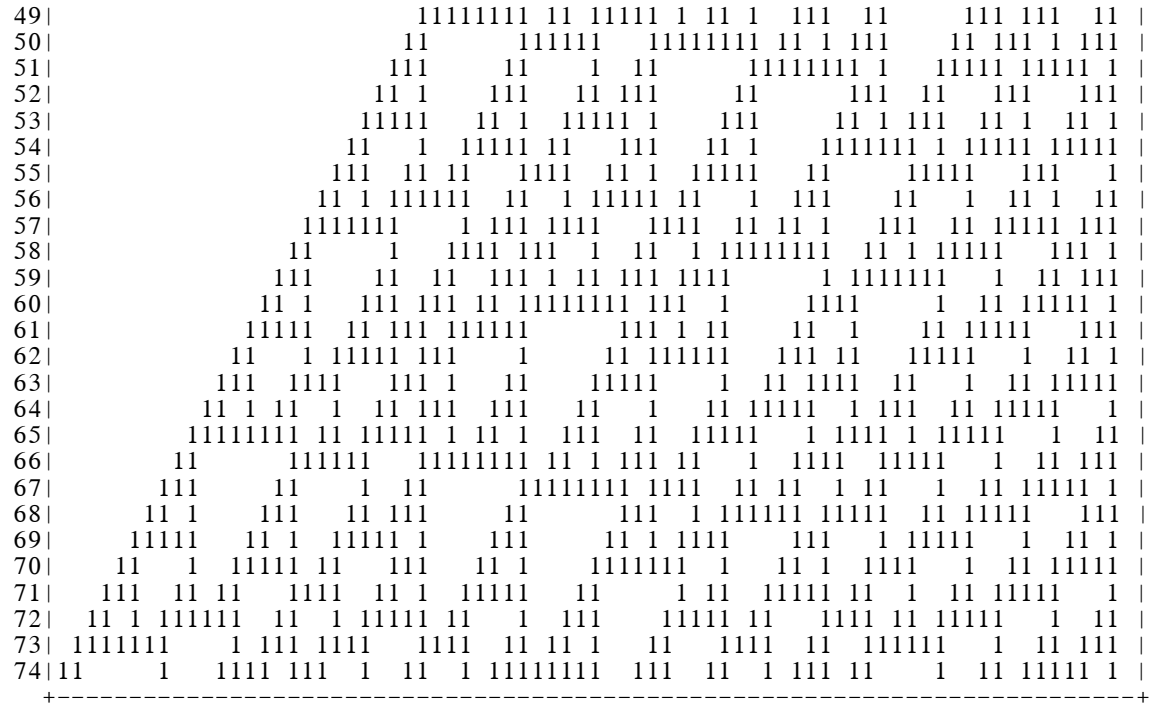


Figure 5. 75 by 75 Final Drawing Output Rendering

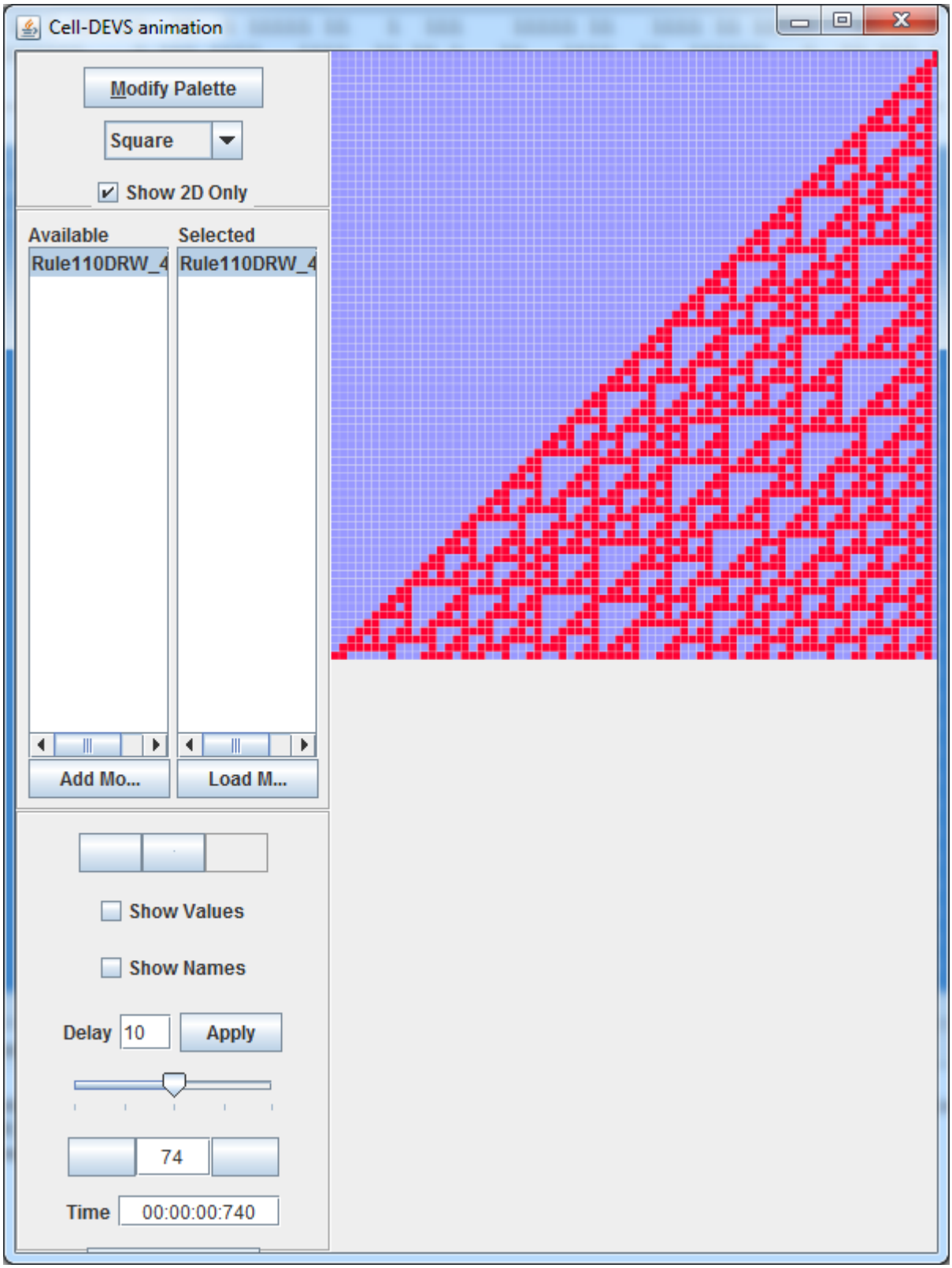


Figure 6. Cell-Devs Animation of a 75 by 75 Interation

Any other input only causes the output to present a “vineyard appearance” as shown in Figure 5. This was generated from a random generation of numbers for the top row by the Boolean Rule Generator for high resolution as discussed above.

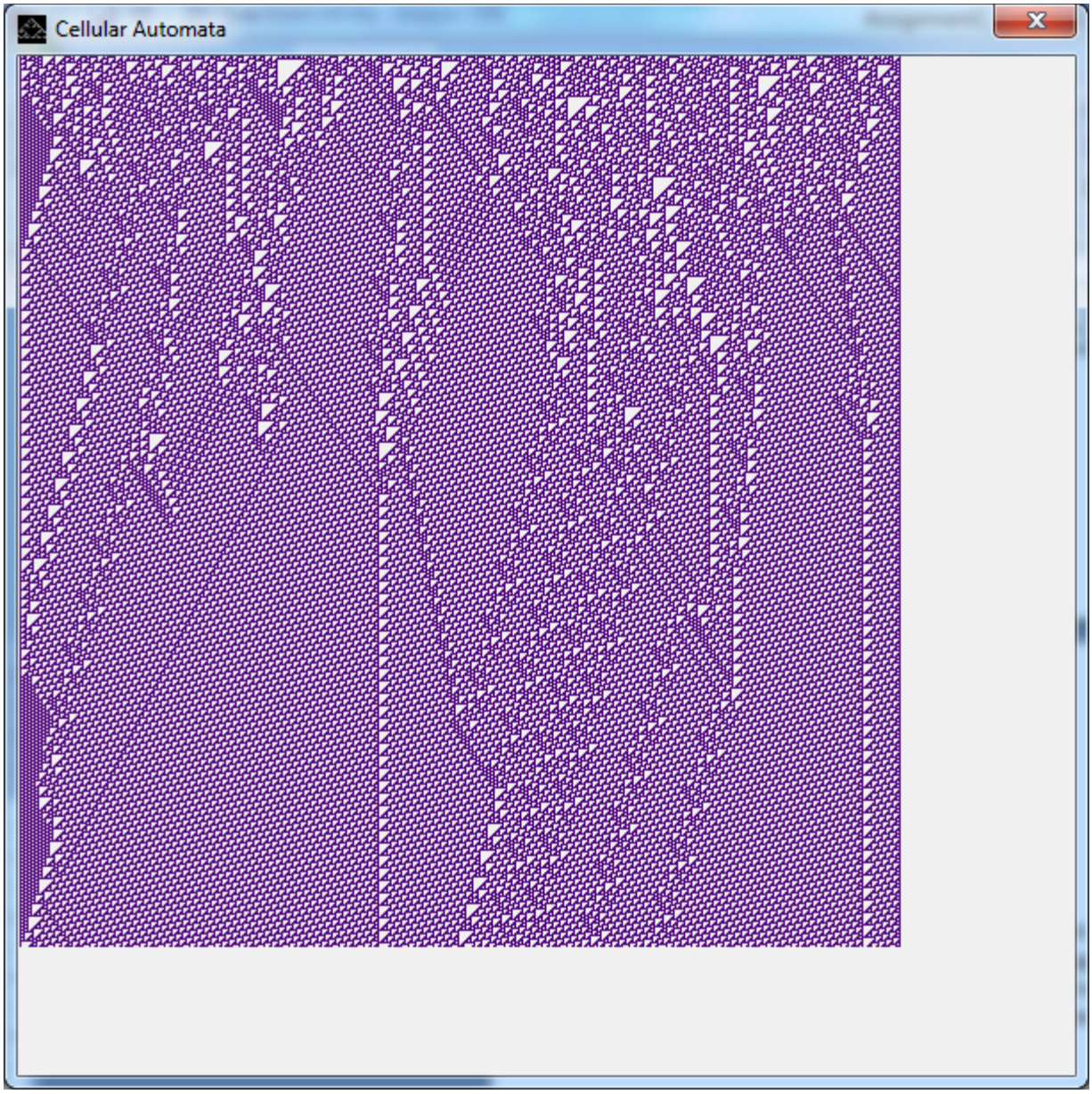


Figure 7. Vineyard Appearance of Rule 110

4.1 Testing Details

The testing files for the Rule 110 Cell-Devs are stored as bat files as is the drawlog executalbe.

4.1.1 30 by 30 Output

The Rule110_30by30.bat file creates a 30 by 30 simulation output while the Rule110_30by30_drw.bat produces the corresponding drawlog output.

The Rule110.val file is set to (0,29) = 1.

The Rule110.ma is dimensioned to dim : (30,30).

The log files required for this bat file are Rule110OUT.out and Rule110LOG.log.

4.1.2 100 by 100 Output

The Rule110_100by100.bat file creates a 100 by 100 simulation output while the Rule110_100by100_drw.bat produces the corresponding drawlog output.

The Rule110.val file is set to (0,99) = 1.

The Rule110.ma is dimensioned to dim : (100,100).

The log files required for this bat file are Rule110OUT_1.out and Rule110LOG_1.log.

4.1.3 200 by 200 Output

The Rule110_200by200.bat file creates a 200 by 200 simulation output while the Rule110_200by200_drw.bat produces the corresponding drawlog output.

The Rule110.val file is set to (0,199) = 1.

The Rule110.ma is dimensioned to dim : (200,200).

The log files required for this bat file are Rule110OUT_2.out and Rule110LOG_2.log.

4.1.3 500 by 500 Output

The Rule110_500by500.bat file creates a 500 by 500 simulation output while the Rule110_500by500_drw.bat produces the corresponding drawlog output.

Note: This will take many hours to process.

The Rule110.val file is set to (0,499) = 1.

The Rule110.ma is dimensioned to dim : (500,500).

The log files required for this bat file are Rule110OUT_3.out and Rule110LOG_3.log.

4.1.4 Boolean Rule Generator

The Boolean Rule Generator executable can generate both a log output comparable to the Cell-Devs drawlog executable and a visual representation as per the Cell-Devs Animator simultaneously. The first row can be manually entered, generated as a random value of 0 or 1 for the number of columns, or a specified right hand value of 1.

The generator has the capability to change the colour of the presentation and clear the display. If this is not desired the generation command button will repopulate the dialog with the last values of the last row of the dialog. This presents the opportunity to change the cellular automata rule within having to enter the data values.

The Boolean Rule generator can generate the output for any of the 256 rules based on the upper neighbours values. As such a cyclic generation command button was added to provide an automatic generation capability. The cyclic generation is based on the rule entered or if blank defaulted to zero and increased. The pattern is then generated and continuously until Rule 255 is reached. The generator is multi-threaded to allow for user interaction with the parent dialog.