# **Discrete Event Modeling and Simulation**

# **SYSC 5104**

# 

# **(2016 Fall)**

**Assignment 2**

**CAVE SYSTEM GENERATION**

**CARLETON UNIVERSITY, OTTAWA**

**PRINCY**

**101055791**

**Part I : Original Conceptual Model**

I propose to use a Cellular-DEVS model to implement a cave generation using cellular automata.

I read an article “Using a cellular automata style rule to create a cave system.” authored by Dana. Email: - **dana@pixelenvy.ca**. I was intrigued by his/her reliable and efficient approach to procedurally generate a system of caves in cellular automata.

According, to the author of this paper, the author took the idea of using cellular automata rules to generate a cave by links from the [**Roguelike Development**](http://www.roguelikedevelopment.org/) site. The concept is simple: a cave is defined as an array of cells, with two states i.e. a wall cell or a floor/empty cell. Here “1” represents that the cell is a wall, and “0” represents that cell is a floor cell (a free cell).

There are two states of cell i.e. wall and the floor. The entire process of cave generation took place in three steps.

The algorithm has three phases:

1. Create the initial map.
2. Run a cycle of CA rules through the map.
3. Join the separate caves together so that any square on the map is reachable by any other.

 Fig1.

Fig 1.depicts some random cave which can be generated using the above algorithm.

To start of an initial map is laid down, for this we start with a rectangular map of cell with a non diggable/permanent border representing the wall cells. Then randomly empty cells are placed in the map. So a clean map is generated to start off with. The second step includes the CA rules which are run for cave generation. The basic rule used for the cave generation is the 4-5 rule which states:

1. For any given square, if it has 3 or fewer adjacent wall squares (counting all 8 cardinal compass points), the square 'starves' and becomes a floor.
2. If it has greater than 5 adjacent wall squares, the square becomes a wall.
3. Otherwise, leave it as is.

CA systems usually run through many cycles and interesting structures can emerge from this.

For cave generation purposes here, though, one cycle was enough to generate an interesting set of caverns. Other CA rules, of course, may produce different cave patterns.

The first two steps of the cave generation algorithm yield a nice looking cave with a few distinct scattered caves. The third step involves joining these caverns into a big, fully connected cave. So for this, think of each cave being a set of floor squares. There are 6-7 disjoint sets one map after step 2. What was needed to do is draw lines of floor squares between the caves. For each square on that line, he union-ed it with one of the sets, stopping when the two sets are union-ed into one (the caves will be joined. At the end he got a nice, clean result showing a 30x30 cave which took 3-4 seconds to generate.

**Part II: Formal Specification**

**A. Cell-DEVS Atomic Model Specification**

The following is the formal specification for the Cell-DEVS cave-generation model:

CD = < X, Y, I, S, θ, N, d, δint, δext, τ, λ, D >

X = Ø

Y = Ø

S = { 0, 1 }

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

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

d = 100 ms

τ: N🡪S is defined by the rules described in the previous section, i.e.:

S = 1 if cell(0,0) = 0 and truecount >5

S = 0 if cell(0,0) = 1 and truecount <4

S= otherwise as it is .

**B. Cell-DEVS Coupled Model Specification**

The model specification from part (A) above gives rise to the following model definition:

[Top]

components : cave generation

[cave\_generation]

type : cell

width : 30

height : 30

delay : inertial

defaultDelayTime : 100

border : wrapped

neighbors : cave\_generation(-1,1) cave\_generation(1,0) cave\_generation(1,1)

neighbors : cave\_generation(0,-1) cave\_generation(0,0) cave\_generation(0,1)

neighbors : cave\_generation(-1,-1) cave\_generation(-1,0) cave\_generation(1,-1)

initialvalue : 0

localtransition : cave\_generation-rule

[cave\_generation-rule]

rule : 1 100 { (0,0) = 0 and trueCount > 5 }

rule : 0 100 { (0,0) = 1 and trueCount < 4 }

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

The model definition also defines the initial random cave by specifying the states of the cells as 1’s or 0’s. For example, the following is a definition of the random cave which can be solved using the Cell-DEVS model:

initialrowvalue : 0 111111111111111111111111111111

initialrowvalue : 1 100000001110111010110110101001

initialrowvalue : 2 111101101001000010111111000001

initialrowvalue : 3 110110111001101110010010001001

initialrowvalue : 4 111100010001101100110110011111

initialrowvalue : 5 110111101110101111000011010111

initialrowvalue : 6 100111110110011011100101000101

initialrowvalue : 7 111110010110010011110101110101

initialrowvalue : 8 101111100101100011101001110011

initialrowvalue : 9 110100111100111110011001000111

initialrowvalue : 10 100111100001011001001011101001

initialrowvalue : 11 100000100111011001010000100111

initialrowvalue : 12 101000001011011110010001001001

initialrowvalue : 13 100000100001010001011101110111

initialrowvalue : 14 111110111001111100001101100001

initialrowvalue : 15 100111111100101011010100010001

initialrowvalue : 16 111101000011010101100101110001

initialrowvalue : 17 101101111010001100101011101101

initialrowvalue : 18 110101010001101000001100110011

initialrowvalue : 19 110101010011010110011110011011

initialrowvalue : 20 111011110010001001101100110101

initialrowvalue : 21 110110010010011111000010101111

initialrowvalue : 22 110111001001010011001110100011

initialrowvalue : 23 100000110000110111011010001111

initialrowvalue : 24 110001100001000100011110101001

initialrowvalue : 25 111101010110001101101100110001

initialrowvalue : 26 110110010010010010111011001011

initialrowvalue : 27 111111011100111000000100111111

initialrowvalue : 28 111110001101011101110010011101

initialrowvalue : 29 111111111111111111111111111111

**C. Implementation and Testing**

The model definitions given above was implemented in one test case with two separate steps, in which initially, a random map was created with 40% of the map being floor. Each step was implemented as a separate model file (called cave1.ma, cave2.ma).

**Step** 1**: A random cave generation.**

The first step was to generate a random cave from 30x30 map of randomly arranged walls and floors. The 30x30 map definition is shown in part (B) above, in the “initialrowvalue” lines of the .ma file. When the model was executed, the rules changed the wall cells with 3 or lesser walls in its neighbor, to a floor and a floor cell with more than 5 wall cells as its neighbors to a wall. This set of rules changed the map to a 30x30 structure with random distinct caves. Therefore, a **step 2** is required with different cellular automata rules and the result of step 1 as the initial values of the cell spaces, to generate a fully connected cave structure. The following diagram illustrates the initial cell state and the random distinct caves, as visualized in the Webviewer tool:

|  |  |
| --- | --- |
|  |  |

**Step 2: Unionizing the random distinct caves.**

Next, the random distinct caves were joined to generate a 30x30 fully connected cave. The model attributes like size and timing were not changed, but new cellular automata rules were used with the initial values of the cell space same as the result of the 1st step. Those are as follow:

initialrowvalue : 0 111111111111111111111111111111

initialrowvalue : 1 111111111111111111111111101001

initialrowvalue : 2 111111111001111111111111000001

initialrowvalue : 3 111111111001111110011110001001

initialrowvalue : 4 111111111001111110000110011111

initialrowvalue : 5 111111111110111111000011001111

initialrowvalue : 6 111111111110011111100011100111

initialrowvalue : 7 111111111110010011110001110011

initialrowvalue : 8 111111111101100011111001110011

initialrowvalue : 9 110111111000111110011001100111

initialrowvalue : 10 100011100001111000000001100011

initialrowvalue : 11 100000000011111000000000100111

initialrowvalue : 12 100000000011111100000001000011

initialrowvalue : 13 110000000001111100001101100011

initialrowvalue : 14 111111111001111100001101100001

initialrowvalue : 15 111111111000111110000100110001

initialrowvalue : 16 111111110000011100000101110001

initialrowvalue : 17 111111110000001100001011111001

initialrowvalue : 18 111111110000001000001100111111

initialrowvalue : 19 111111110000000100011110011111

initialrowvalue : 20 111111110000001111101100011111

initialrowvalue : 21 111111100000011111000110001111

initialrowvalue : 22 110111000000011111001110000111

initialrowvalue : 23 100000100000000111111110000111

initialrowvalue : 24 110001100000000111111110000001

initialrowvalue : 25 111101000000001101111111110001

initialrowvalue : 26 111110011000010000111111111011

initialrowvalue : 27 111111111100111000111111111111

initialrowvalue : 28 111111111111111101111111111111

initialrowvalue : 29 111111111111111111111111111111

When this model was executed, the result was that all the distinct caves were joined to each other and a fully connected 30x30 fully connected cave was generated in 1.2 secs. The following diagram illustrates the initial cell state and the final steady, as visualized in the Graflog tool:

|  |  |
| --- | --- |
|  |  |

**Part III: Conclusions**

The Cellular-DEVS models included with this report and described herein correctly simulate the behavior of the cave system generation algorithm by Dana in her article. The caves, modeled as cell spaces, can be generated by destroying the old or creating new floor and wall cells, in order to fill bigger spaces with floor cell. This algorithm has an advantage over other cave generating algorithms as it can fully connected cave system instead of just leaving it with random distinct caves.

The downside to this algorithm is that it sometimes doesn’t join all the distinct cave, but can be made possible by increasing the no. of runs. Even it won’t work as the same for the cave in squares bigger than 50x50.

The logic behind the algorithm is simple but a bit tricky. It would be interesting, in further studies, to determine if this algorithm would scale to 3-dimensional spaces.

This assignment was also successful in demonstrating the use of the CD++ tool in simulating a cellular automata model, and using the associated tools (CD++ Builder, Webviewer) to visualize the outputs.