**SYSC 5104: Methodologies for Discrete-Event Modeling and Simulation**

**Instructor: Prof. Gabriel A. Wainer**

**Generation of Chess Variants**

**Assignment 2**

**Prepared By: Ritvik Rajnikant Pandey**

**Student Id: 101090052**

**Part I: Generation of Chess Variants**

Ever since Conway’s Game of Life was introduced, the interest for cellular automata has constantly increased, yielding interesting results and applications within many areas, ranging from pattern generation in arts such as music composition to less obvious applications such as language recognition.

In software system described in the reference paper a number of artificial intelligence systems were integrated to generate a computer chess variant AI. A chess variant is known as a game that is a variation of regular chess with respect to mainly the set of rules, pieces-sets or board configurations. The aim of this reference paper is the presentation of one of these subsystems, using modular binary 2D cellular automata based on a new set of rules, or more specifically a new variation of the game of life to generate patterns that could be used as design elements in applications such as computer games or user interfaces.

In this work five cellular automata, each consisting of a new version of Conway’s Game of Life were implemented and tested. Only rule 1 was implemented in the game, which showed to be sufficient for the generation of the chess variants. The rule is formulated below:

**Rule: A cell is set to 1 if it has 2-3 neighbors else to 0.**

After implementing the rule, the second step followed is the evolution of space for these automata on a large and small grid. Since the same pattern must sooner or later emerge in any finite binary grid, such system will eventually either converge into a static configuration or a cyclic loop where a number of patterns, by definition equal or larger than two, are endlessly repeated. In the third step a variety of kernels can be used to initiate the cellular automata however, only two of them were used for implementation.

**Part II: Formal Specification**

**Cell-DEVS Atomic Model Specification**

The following is the formal specification for the Cell-DEVS generation of chess variants 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), (-1,0), (-1,1), (0, -1), (0,0), (0,1), (1, -1), (1,0), (1,1)}

d = 100 ms

τ: N🡪S is defined by the following rules:

S = 1 if cell(0,0) = 1 and (trueCount = 3 or trueCount = 4)

S = 1 if cell(0,0) = 0 and (trueCount = 2 or trueCount = 3)

S = 0 for any other condition

**Cell-DEVS Coupled Model Specification for chessVariantKernel1**

[top]

components : chessVariantKernel1

[chessVariantKernel1]

type : cell

width : 13

height : 9

delay : transport

defaultDelayTime : 100

border : wrapped

neighbors : chessVariantKernel1(-1,-1) chessVariantKernel1(-1,0) chessVariantKernel1(-1,1)

neighbors : chessVariantKernel1(0,-1) chessVariantKernel1(0,0) chessVariantKernel1(0,1)

neighbors : chessVariantKernel1(1,-1) chessVariantKernel1(1,0) chessVariantKernel1(1,1)

initialvalue : 0

initialCellsValue : chessVariantKernel1.val

localtransition : chessVariantKernel1-rule

[chessVariantKernel1-rule]

rule : 1 100 { (0,0) = 1 and ( trueCount = 3 or trueCount = 4 ) }

rule : 1 100 { (0,0) = 0 and ( trueCount = 2 or trueCount = 3 ) }

rule : 0 100 { t }

**Cell-DEVS Coupled Model Specification for chessVariantKernel2**

[top]

components : chessVariantKernel2

[chessVariantKernel2]

type : cell

width : 13

height : 9

delay : transport

defaultDelayTime : 100

border : wrapped

neighbors : chessVariantKernel2(-1,-1) chessVariantKernel2(-1,0) chessVariantKernel2(-1,1)

neighbors : chessVariantKernel2(0,-1) chessVariantKernel2(0,0) chessVariantKernel2(0,1)

neighbors : chessVariantKernel2(1,-1) chessVariantKernel2(1,0) chessVariantKernel2(1,1)

initialvalue : 0

initialCellsValue : chessVariantKernel2.val

localtransition : chessVariantKernel2-rule

[chessVariantKernel2-rule]

rule : 1 100 { (0,0) = 1 and ( trueCount = 3 or trueCount = 4 ) }

rule : 1 100 { (0,0) = 0 and ( trueCount = 2 or trueCount = 3 ) }

rule : 0 100 { t }

**Cell-DEVS Coupled Model Specification for chessVariantKernel1SmallGrid**

[top]

components : chessVariantKernel1SmallGrid

[chessVariantKernel1SmallGrid]

type : cell

width : 7

height : 7

delay : transport

defaultDelayTime : 100

border : wrapped

neighbors : chessVariantKernel1SmallGrid(-1,-1) chessVariantKernel1SmallGrid(-1,0) chessVariantKernel1SmallGrid(-1,1)

neighbors : chessVariantKernel1SmallGrid(0,-1) chessVariantKernel1SmallGrid(0,0) chessVariantKernel1SmallGrid(0,1)

neighbors : chessVariantKernel1SmallGrid(1,-1) chessVariantKernel1SmallGrid(1,0) chessVariantKernel1SmallGrid(1,1)

initialvalue : 0

initialCellsValue : chessVariantKernel1SmallGrid.val

localtransition : chessVariantKernel1SmallGrid-rule

[chessVariantKernel1SmallGrid-rule]

rule : 1 100 { (0,0) = 1 and ( trueCount = 3 or trueCount = 4 ) }

rule : 1 100 { (0,0) = 0 and ( trueCount = 2 or trueCount = 3 ) }

rule : 0 100 { t }

**Cell-DEVS Coupled Model Specification for chessVariantKernel2SmallGrid**

[top]

components : chessVariantKernel2SmallGrid

[chessVariantKernel2SmallGrid]

type : cell

width : 9

height : 5

delay : transport

defaultDelayTime : 100

border : wrapped

neighbors : chessVariantKernel2SmallGrid(-1,-1) chessVariantKernel2SmallGrid(-1,0) chessVariantKernel2SmallGrid(-1,1)

neighbors : chessVariantKernel2SmallGrid(0,-1) chessVariantKernel2SmallGrid(0,0) chessVariantKernel2SmallGrid(0,1)

neighbors : chessVariantKernel2SmallGrid(1,-1) chessVariantKernel2SmallGrid(1,0) chessVariantKernel2SmallGrid(1,1)

initialvalue : 0

initialCellsValue : chessVariantKernel2SmallGrid.val

localtransition : chessVariantKernel2SmallGrid-rule

[chessVariantKernel2SmallGrid-rule]

rule : 1 100 { (0,0) = 1 and ( trueCount = 3 or trueCount = 4 ) }

rule : 1 100 { (0,0) = 0 and ( trueCount = 2 or trueCount = 3 ) }

rule : 0 100 { t }

**The Initial Values for chessVariantKernel1 model:**

|  |  |  |
| --- | --- | --- |
| (0,0) = 0  (0,1) = 0  (0,2) = 0  (0,3) = 0  (0,4) = 0  (0,5) = 0  (0,6) = 0  (0,7) = 0  (0,8) = 0  (0,9) = 0  (0,10) = 0  (0,11) = 0  (0,12) = 0 | (3,0) = 0  (3,1) = 0  (3,2) = 0  (3,3) = 0  (3,4) = 0  (3,5) = 0  (3,6) = 1  (3,7) = 0  (3,8) = 0  (3,9) = 0  (3,10) = 0  (3,11) = 0  (3,12) = 0 | (6,0) = 0  (6,1) = 0  (6,2) = 0  (6,3) = 0  (6,4) = 0  (6,5) = 0  (6,6) = 0  (6,7) = 0  (6,8) = 0  (6,9) = 0  (6,10) = 0  (6,11) = 0  (6,12) = 0 |
| (1,0) = 0  (1,1) = 0  (1,2) = 0  (1,3) = 0  (1,4) = 0  (1,5) = 0  (1,6) = 0  (1,7) = 0  (1,8) = 0  (1,9) = 0  (1,10) = 0  (1,11) = 0  (1,12) = 0 | (4,0) = 0  (4,1) = 0  (4,2) = 0  (4,3) = 0  (4,4) = 0  (4,5) = 1  (4,6) = 1  (4,7) = 1  (4,8) = 0  (4,9) = 0  (4,10) = 0  (4,11) = 0  (4,12) = 0 | (7,0) = 0  (7,1) = 0  (7,2) = 0  (7,3) = 0  (7,4) = 0  (7,5) = 0  (7,6) = 0  (7,7) = 0  (7,8) = 0  (7,9) = 0  (7,10) = 0  (7,11) = 0  (7,12) = 0 |
| (2,0) = 0  (2,1) = 0  (2,2) = 0  (2,3) = 0  (2,4) = 0  (2,5) = 0  (2,6) = 0  (2,7) = 0  (2,8) = 0  (2,9) = 0  (2,10) = 0  (2,11) = 0  (2,12) = 0 | (5,0) = 0  (5,1) = 0  (5,2) = 0  (5,3) = 0  (5,4) = 0  (5,5) = 0  (5,6) = 1  (5,7) = 0  (5,8) = 0  (5,9) = 0  (5,10) = 0  (5,11) = 0  (5,12) = 0 | (8,0) = 0  (8,1) = 0  (8,2) = 0  (8,3) = 0  (8,4) = 0  (8,5) = 0  (8,6) = 0  (8,7) = 0  (8,8) = 0  (8,9) = 0  (8,10) = 0  (8,11) = 0  (8,12) = 0 |

**The Initial Values for chessVariantKernel2 model:**

|  |  |  |
| --- | --- | --- |
| (0,0) = 0  (0,1) = 0  (0,2) = 0  (0,3) = 0  (0,4) = 0  (0,5) = 0  (0,6) = 0  (0,7) = 0  (0,8) = 0  (0,9) = 0  (0,10) = 0  (0,11) = 0  (0,12) = 0 | (3,0) = 0  (3,1) = 0  (3,2) = 0  (3,3) = 0  (3,4) = 0  (3,5) = 0  (3,6) = 1  (3,7) = 1  (3,8) = 0  (3,9) = 0  (3,10) = 0  (3,11) = 0  (3,12) = 0 | (6,0) = 0  (6,1) = 0  (6,2) = 0  (6,3) = 0  (6,4) = 0  (6,5) = 0  (6,6) = 0  (6,7) = 0  (6,8) = 0  (6,9) = 0  (6,10) = 0  (6,11) = 0  (6,12) = 0 |
| (1,0) = 0  (1,1) = 0  (1,2) = 0  (1,3) = 0  (1,4) = 0  (1,5) = 0  (1,6) = 0  (1,7) = 0  (1,8) = 0  (1,9) = 0  (1,10) = 0  (1,11) = 0  (1,12) = 0 | (4,0) = 0  (4,1) = 0  (4,2) = 0  (4,3) = 0  (4,4) = 0  (4,5) = 1  (4,6) = 1  (4,7) = 1  (4,8) = 0  (4,9) = 0  (4,10) = 0  (4,11) = 0  (4,12) = 0 | (7,0) = 0  (7,1) = 0  (7,2) = 0  (7,3) = 0  (7,4) = 0  (7,5) = 0  (7,6) = 0  (7,7) = 0  (7,8) = 0  (7,9) = 0  (7,10) = 0  (7,11) = 0  (7,12) = 0 |
| (2,0) = 0  (2,1) = 0  (2,2) = 0  (2,3) = 0  (2,4) = 0  (2,5) = 0  (2,6) = 0  (2,7) = 0  (2,8) = 0  (2,9) = 0  (2,10) = 0  (2,11) = 0  (2,12) = 0 | (5,0) = 0  (5,1) = 0  (5,2) = 0  (5,3) = 0  (5,4) = 0  (5,5) = 1  (5,6) = 1  (5,7) = 0  (5,8) = 0  (5,9) = 0  (5,10) = 0  (5,11) = 0  (5,12) = 0 | (8,0) = 0  (8,1) = 0  (8,2) = 0  (8,3) = 0  (8,4) = 0  (8,5) = 0  (8,6) = 0  (8,7) = 0  (8,8) = 0  (8,9) = 0  (8,10) = 0  (8,11) = 0  (8,12) = 0 |

**The Initial Values for chessVariantKernel1SmallGrid model:**

|  |  |  |
| --- | --- | --- |
| (0,0) = 0  (0,1) = 0  (0,2) = 0  (0,3) = 0  (0,4) = 0  (0,5) = 0  (0,6) = 0 | (3,0) = 0  (3,1) = 0  (3,2) = 1  (3,3) = 1  (3,4) = 1  (3,5) = 0  (3,6) = 0 | (6,0) = 0  (6,1) = 0  (6,2) = 0  (6,3) = 0  (6,4) = 0  (6,5) = 0  (6,6) = 0 |
| (1,0) = 0  (1,1) = 0  (1,2) = 0  (1,3) = 0  (1,4) = 0  (1,5) = 0  (1,6) = 0 | (4,0) = 0  (4,1) = 0  (4,2) = 0  (4,3) = 1  (4,4) = 0  (4,5) = 0  (4,6) = 0 |  |
| (2,0) = 0  (2,1) = 0  (2,2) = 0  (2,3) = 1  (2,4) = 0  (2,5) = 0  (2,6) = 0 | (5,0) = 0  (5,1) = 0  (5,2) = 0  (5,3) = 0  (5,4) = 0  (5,5) = 0  (5,6) = 0 |  |

**The Initial Values for chessVariantKernel2SmallGrid model:**

|  |  |  |
| --- | --- | --- |
| (0,0) = 0  (0,1) = 0  (0,2) = 0  (0,3) = 0  (0,4) = 0  (0,5) = 0  (0,6) = 0  (0,7) = 0  (0,8) = 0 | (2,0) = 0  (2,1) = 0  (2,2) = 0  (2,3) = 1  (2,4) = 1  (2,5) = 1  (2,6) = 0  (2,7) = 0  (2,8) = 0 | (4,0) = 0  (4,1) = 0  (4,2) = 0  (4,3) = 0  (4,4) = 0  (4,5) = 0  (4,6) = 0  (4,7) = 0  (4,8) = 0 |
| (1,0) = 0  (1,1) = 0  (1,2) = 0  (1,3) = 0  (1,4) = 1  (1,5) = 1  (1,6) = 0  (1,7) = 0  (1,8) = 0 | (3,0) = 0  (3,1) = 0  (3,2) = 0  (3,3) = 1  (3,4) = 1  (3,5) = 0  (3,6) = 0  (3,7) = 0  (3,8) = 0 |  |

**Implementation and Testing**

A variety of kernels can be used to initiate the cellular automata. However, two kernels were used to generate the patterns out of which one was a cross shaped and the other was an asymmetric one. The cellular automata rules were applied on both the kernels and were tested on a small and a large grid size to generate the patterns and observe the behavior.

**Model 1:**

This model displays the chessVariantKernel1 on a 13 x 9 grid. The cross shaped kernel was tested on a large grid which displayed how the kernel evolves over 0 to 255 generations. The result for this can be better viewed in the simulation video.

**Model 2:**

The model displays the chessVariantKernel2 on a 13 x 9 grid. The asymmetric kernel was tested on a large grid which displayed the evolution of the kernel until 136th generation and then apparently falling into a cycle. The results can be better viewed in the simulation video.

**Model 3:**

The model displays the chessVariantKernel1SmallGrid on a 7 x 7 grid. The cross shaped kernel is tested again on a small grid. The kernel falls into a cycle after the 4th generation, the results can be seen below and also in the simulation video.

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

**Model 4:**

The model displays the chessVariantKernel2SmallGrid on a 9 x 5 grid. The asymmetric kernel is tested on a smaller grid where it becomes symmetric after the 12th generation and eventually falls into a cycle after the 38th generation. The simulation results can be better viewed in the simulation video.

**All four models were tested as per given in the reference paper and were found to be true.**

**References:**

[1] Mikael Fridenfalk : Application of Cellular Automata for Generation of Chess Variants