SYSC5104: Methodologies in Discrete Event Modeling and Simulation

Assignment 2:

Nearest Neighbour Classifier

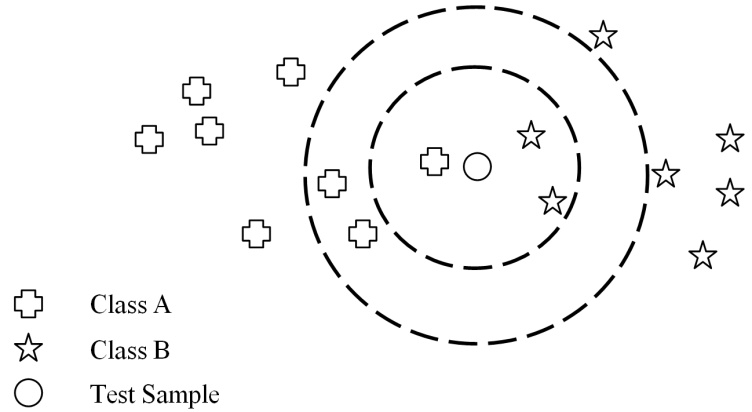
Seyed Ali Etemad

Student Number: 100333627

Professor: Gabriel Wainer

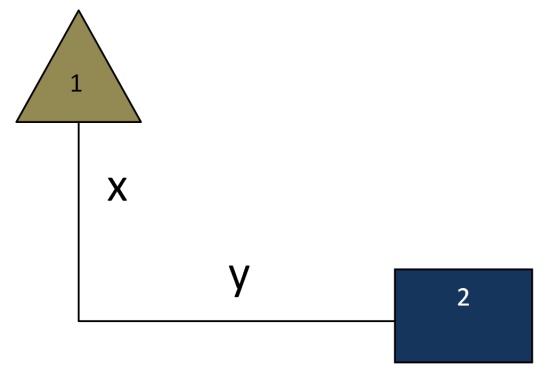
**Part I**

K-Nearest Neighbour (KNN) is one of the most popular types of classifiers often used for pattern classification and data mining. Different definitions for measuring distance, especially Euclidean distance, formulate the basic concept of KNN. KNN is a supervised classification technique which classifies test data based on the *K* number of closest training samples (where *K* is a positive integer) to the test sample. Figure 1 represents the basic concept of KNN where the test sample has been classified as class B for K=2 and as class A for K=3.



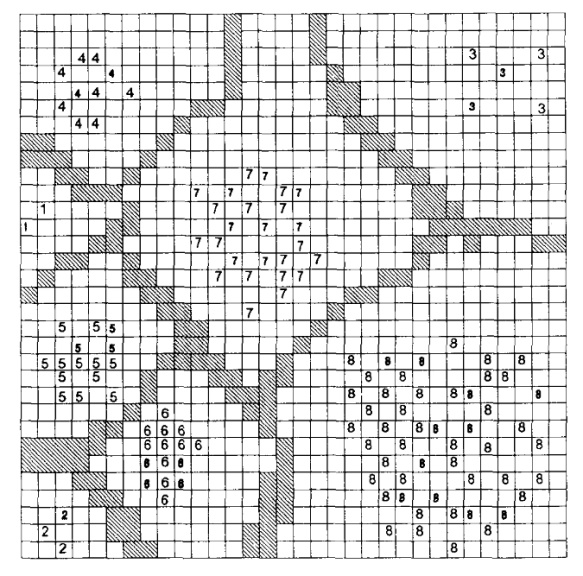
**Fig 1. Different KNN classification results with K=2 and K=3**

In the case of K=1, the K-nearest neighbour classifier is called the *Nearest Neighbour* classifier. This means that unknown data are labelled based on the closest training sample. The nearest neighbour classifier is considered a very simple yet handy tool for pattern recognition purposes. Different means of measuring distance are available, for instance the Euclidean distance or Manhatan distance. In this assignment we have chosen the Manhatan distance which can be accurately modeled by defining the Von-Neumann neighbourhood. Figure 2 presents the Manhatan distance of two objects.



**Fig 2. Manhatan distance between object 1 and 2 is *( x + y )***

At instances where the distance between two test samples is the exactly equal, discriminant curves will be formed. This will occur when the distance between two samples is an odd number of unit cells. For instances where the distance between the samples is an even number of cells, the two cells forming the border will form the discriminant curve and result in a thicker curve. Figure 3 illustrates the division of the feature space based on different samples. The discriminant cures for odd and even distance cells are also provided in Figure 3.



**Fig 3. Discriminant curves, directly reproduced from [1]. The red circles show the single and double curves (odd and even spacing between samples)**

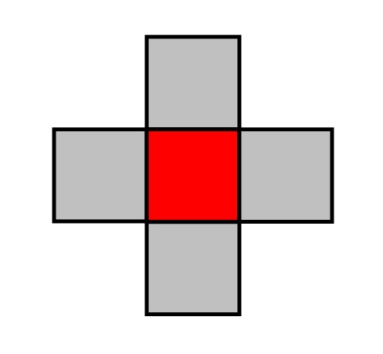
Since Cell-DEVS provide a very suitable space for modeling events related to and defined by distance, KNN can be accurately defined and modelled in this environment. Various number and formation of training samples are tested and respective discriminant curves are derived.

The reference used for this assignment is [1]. In this paper, however, the authors have not mentioned the exact rules used to manipulate the cell-space and have referred to [2], thus the rules were derived. The results obtained by this method are identical to those of [1].

**Part II**

**Neighborhood**

In pattern recognition applications, for 2D data and in Euclidean space, the neighbourhood is usually defined as a circle around the sample. For Manhatan distance definition and in cell-space however, it is very logical to define the neighbourhood as the Von-Neumann neighbourhood, since diagonal lines are not defined. If the usual 3 x 3 neighbourhood was adapted, we would be considering diagonal distances as well, which we want to avoid. Figure 4 illustrates the defined neighbourhood.



**Fig 4. The defined neighbourhood**

**Algorithm**

The following is a set of rules used throughout this assignment.

1. Empty cells are initialized as 0.
2. The training samples are labelled from 101, 102, 103, 104, 105, 106, and 107 for each sample (7 classes overall).
3. Each cell emits a decimal value IF the center of the neighbourhood is on an empty cell AND it is in contact with a class sample OR another decimal value equal to itself. This will result in the growth of all class boundaries.

rule : 0.1 25 { (0,0) = 0 and (((stateCount(101) = 1 or stateCount(0.1) > 0) and stateCount(0) > 2) or stateCount(101) > 1) }

rule : 0.2 25 { (0,0) = 0 and (((stateCount(102) = 1 or stateCount(0.2) > 0) and stateCount(0) > 2) or stateCount(102) > 1) }

rule : 0.3 25 { (0,0) = 0 and (((stateCount(103) = 1 or stateCount(0.3) > 0) and stateCount(0) > 2) or stateCount(103) > 1) }

rule : 0.4 25 { (0,0) = 0 and (((stateCount(104) = 1 or stateCount(0.4) > 0) and stateCount(0) > 2) or stateCount(104) > 1) }

rule : 0.5 25 { (0,0) = 0 and (((stateCount(105) = 1 or stateCount(0.5) > 0) and stateCount(0) > 2) or stateCount(105) > 1) }

rule : 0.6 25 { (0,0) = 0 and (((stateCount(106) = 1 or stateCount(0.6) > 0) and stateCount(0) > 2) or stateCount(106) > 1) }

rule : 0.7 25 { (0,0) = 0 and (((stateCount(107) = 1 or stateCount(0.7) > 0) and stateCount(0) > 2) or stateCount(107) > 1) }

1. In any case where an empty cell in the center of a neighbourhood is in contact with 2 or less empty cells (2, 1 or 0), that cell is given the value 100 which denotes a cell of a discriminant curve. The reason is that this rule indicates that the empty cell is surrounded by another cell of decimal value (odd distance).

rule : 100 25 { (0,0) = 0 and stateCount(0) <= 2 }

1. In any case where the center cell is an emitted cell itself yet it is contact with another emitted cell of a different value, it should be assigned the value 100 (curve cell). The reason for this is that at instances where we have an even distance between two samples, the emitted values will fill up the space up to a point where two cells remain. The remaining two cells are then filled, each by its closest value. Based on Figure 3 however, these boundary points should form the thick double boundary discriminant curve.

rule : 100 25 { (0,0) = 0.1 and stateCount(0) + stateCount(0.1) + stateCount(100) + stateCount(101) != 5 }

rule : 100 25 { (0,0) = 0.2 and stateCount(0) + stateCount(0.2) + stateCount(100) + stateCount(102) != 5 }

rule : 100 25 { (0,0) = 0.3 and stateCount(0) + stateCount(0.3) + stateCount(100) + stateCount(103) != 5 }

rule : 100 25 { (0,0) = 0.4 and stateCount(0) + stateCount(0.4) + stateCount(100) + stateCount(104) != 5 }

rule : 100 25 { (0,0) = 0.5 and stateCount(0) + stateCount(0.5) + stateCount(100) + stateCount(105) != 5 }

rule : 100 25 { (0,0) = 0.6 and stateCount(0) + stateCount(0.6) + stateCount(100) + stateCount(106) != 5 }

rule : 100 25 { (0,0) = 0.7 and stateCount(0) + stateCount(0.7) + stateCount(100) + stateCount(107) != 5 }

1. Cells which are not of the training samples, and are in contact with 4 emitted cells of the same kind OR, are in contact with 3 emitted cells of the same kind and one empty cell, must be labelled as its non-zero surrounding cells. This is for special cases (modification of the curve cells).

rule : 0.1 25 { (0,0) != 101 and (stateCount(0.1) = 4 or (stateCount(0.1) = 3 and stateCount(0) = 1) ) }

rule : 0.2 25 { (0,0) != 102 and (stateCount(0.2) = 4 or (stateCount(0.2) = 3 and stateCount(0) = 1) ) }

rule : 0.3 25 { (0,0) != 103 and (stateCount(0.3) = 4 or (stateCount(0.3) = 3 and stateCount(0) = 1) ) }

rule : 0.4 25 { (0,0) != 104 and (stateCount(0.4) = 4 or (stateCount(0.4) = 3 and stateCount(0) = 1) ) }

rule : 0.5 25 { (0,0) != 105 and (stateCount(0.5) = 4 or (stateCount(0.5) = 3 and stateCount(0) = 1) ) }

rule : 0.6 25 { (0,0) != 106 and (stateCount(0.6) = 4 or (stateCount(0.6) = 3 and stateCount(0) = 1) ) }

rule : 0.7 25 { (0,0) != 107 and (stateCount(0.7) = 4 or (stateCount(0.7) = 3 and stateCount(0) = 1) ) }

1. Leave other cells as they are.

rule : { (0,0) } 25 { t }

**DEVS Formulism**

In this assignment, external coupling or an outside world, thus there is no need to define M.

M = < X, Y, D, {Mi}, {Ii}, {Zij}, select>

**Atomic Cell-DEVS Model**

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

X: Cell sees its following neighbourhood as the inputs to evaluate its next state:

knn(-1,0)

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

knn(1,0)

Y: No output is available from a cell. Cell state is only updated based on neighbours.

S = {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 100, 101, 102, 103, 104, 105, 106, 107}

N = {N/A} (There are no input events for this model)

I = < n, u, Px, Py > = {N/A}

θ = (s, phase, σqueue, σ) Each state change occurs after the transport delay.

δint , δext , τ, λ, D = N/A (These functions are not available)

## Couple Cell-DEVS

GCC = < XList, YList, I, X, Y, η, N, {r, c}, C, B, Z, select >

XList = { (k,l,) where k and l belongs to N, 0 ≤ k ≤ 31, 0 ≤ l ≤ 31}

YList = { N/A }

I = <Px, Py> = { N/A }

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

X = { N/A }

Y = { N/A }

η = 5

r = 31

c = 31

C = { Cij/ i ϵ [0,31], j ϵ [0,31]}, where Cij = < Iij, Xij, Yij, Sij, Nij, dij, δintij, δextij, τij, λij, Dij> is an atomic component.

B = B = {Cij / (i = 1 v i = f) ʌ (j = 1 v j = c) ʌ Cij ϵ C ʌ Cij = < Iij, *Xij*, *Yij*, Sij, Nij, dij, δintij, δextij, dintij, dextij, tij, lij, Dij> is an atomic cell}, if the border has different behavior than the rest of the cell space.

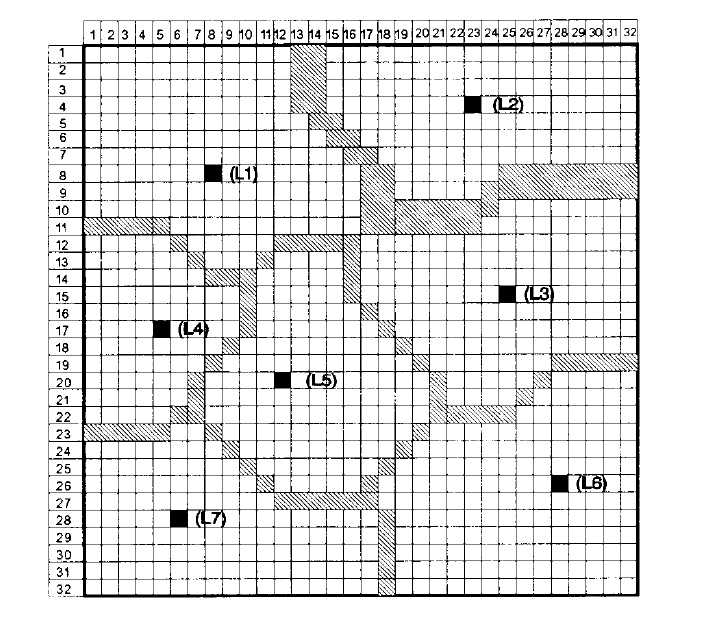
Z = N/A (no coupling)

Select = N/A (no priority)

**Testing and Animation**

Using the Modeler, 3 of the cases used in [1] are tested.

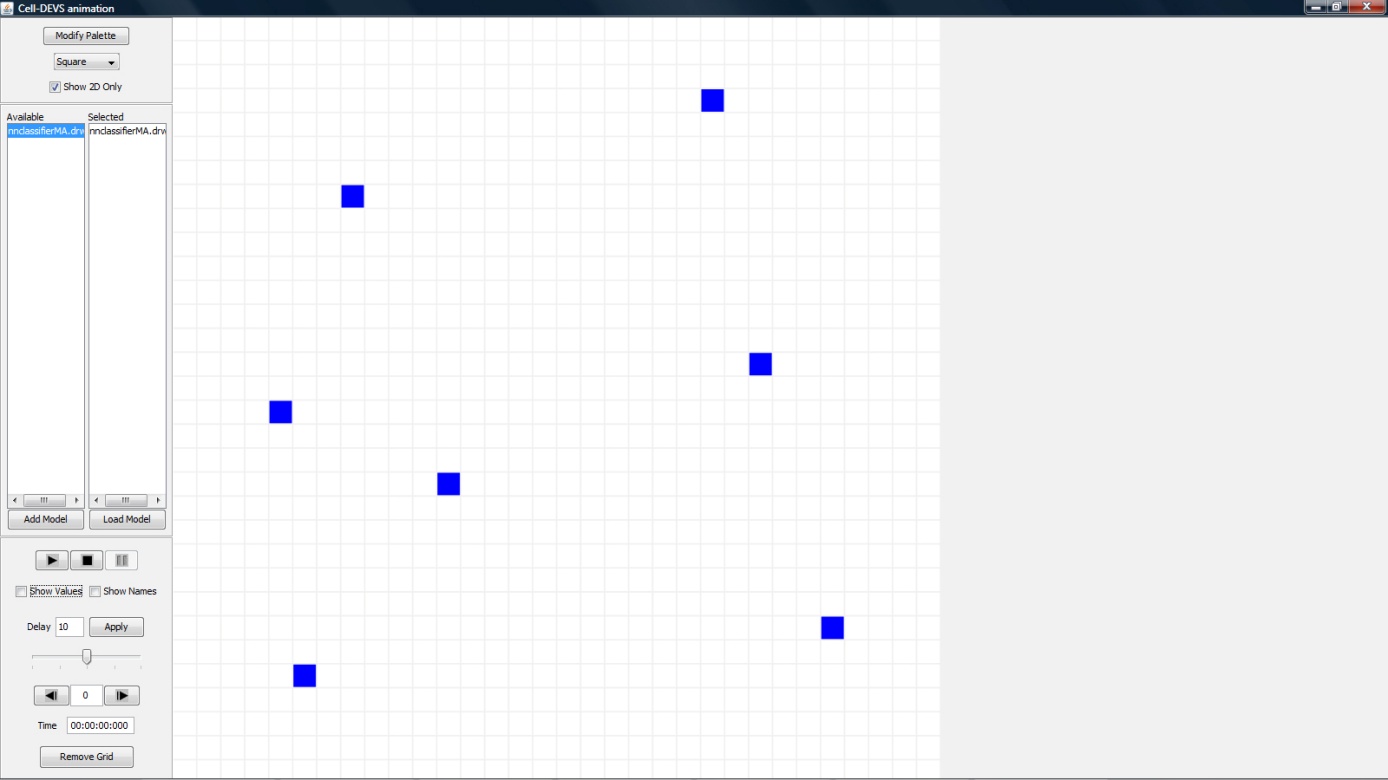
Figure 5 is from [1] where the authors have divided the feature space based on training samples. Discriminant curves are also presented. This is modeled by the initial value file “nn1.val”.

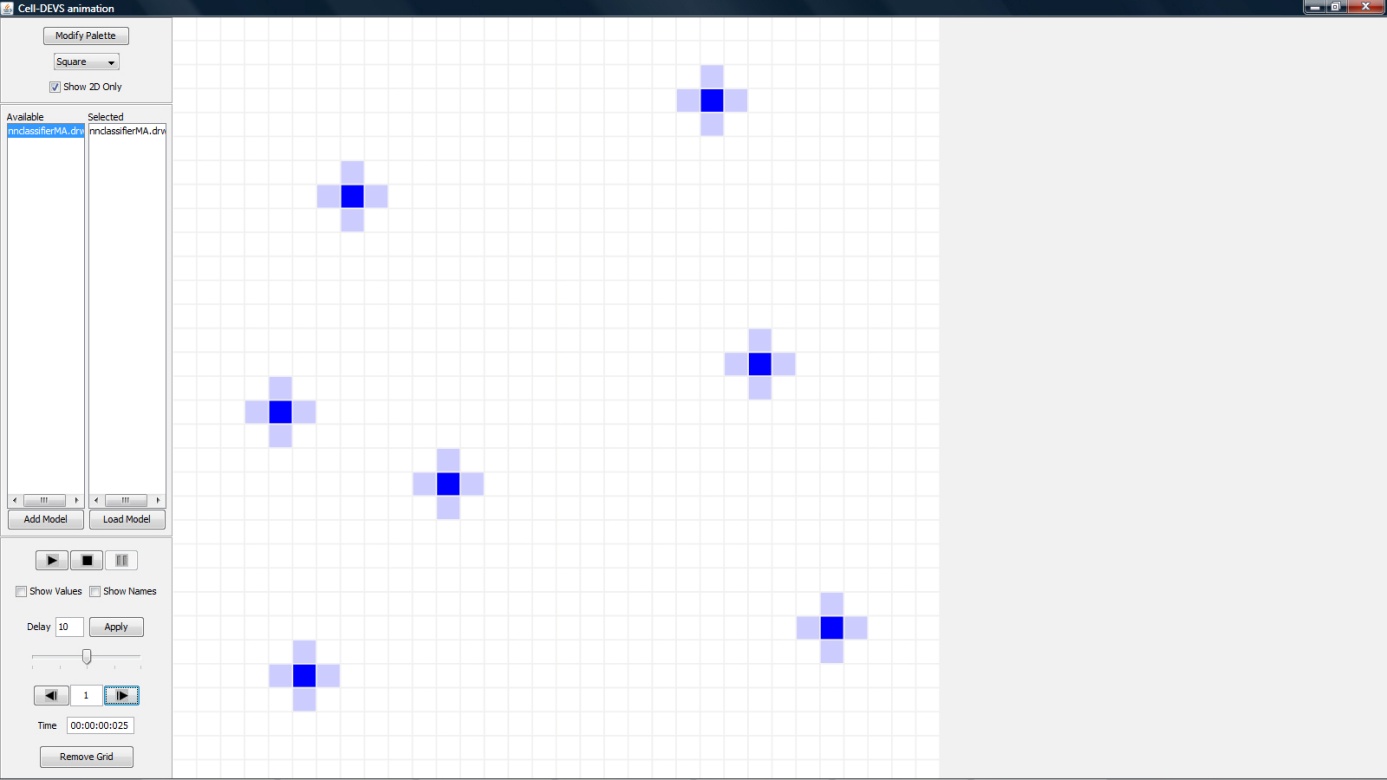


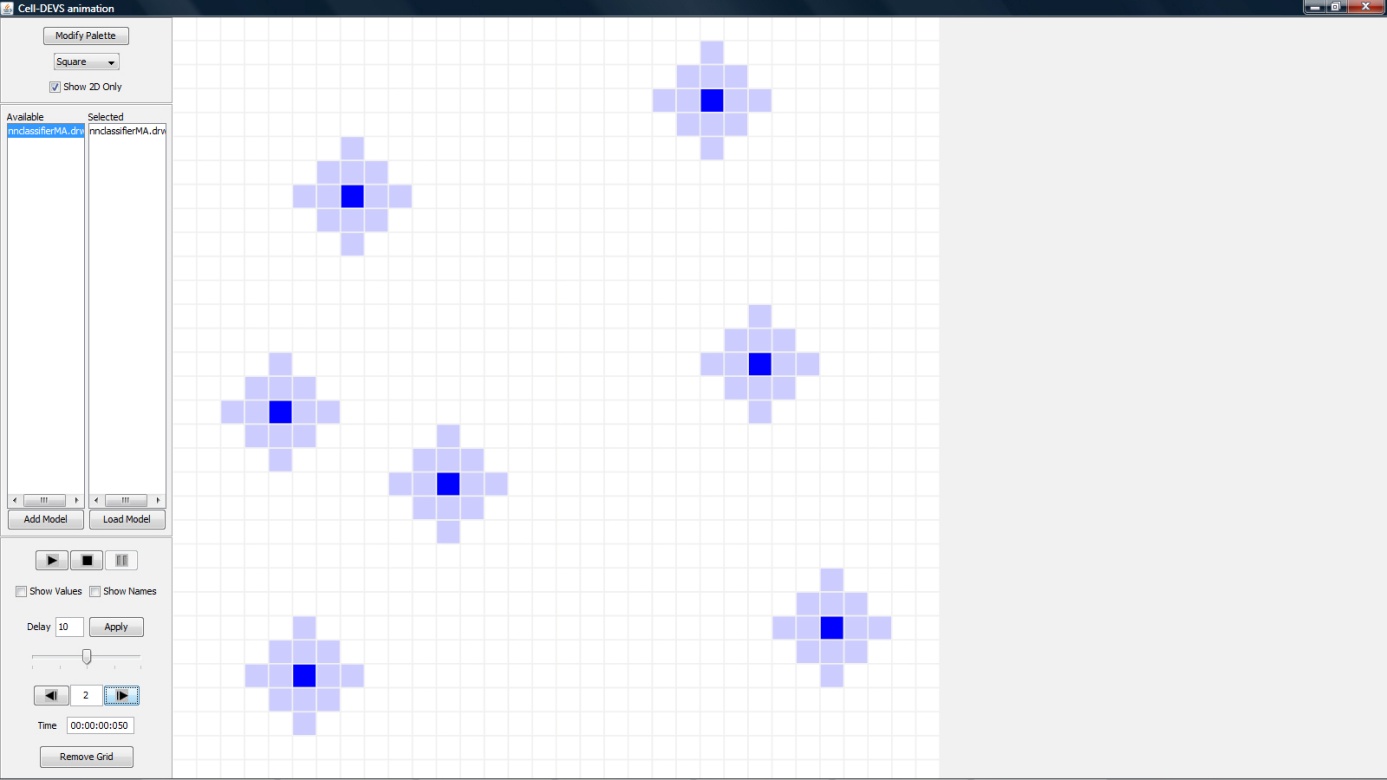
**Fig 4. Directly reproduced from [1]**

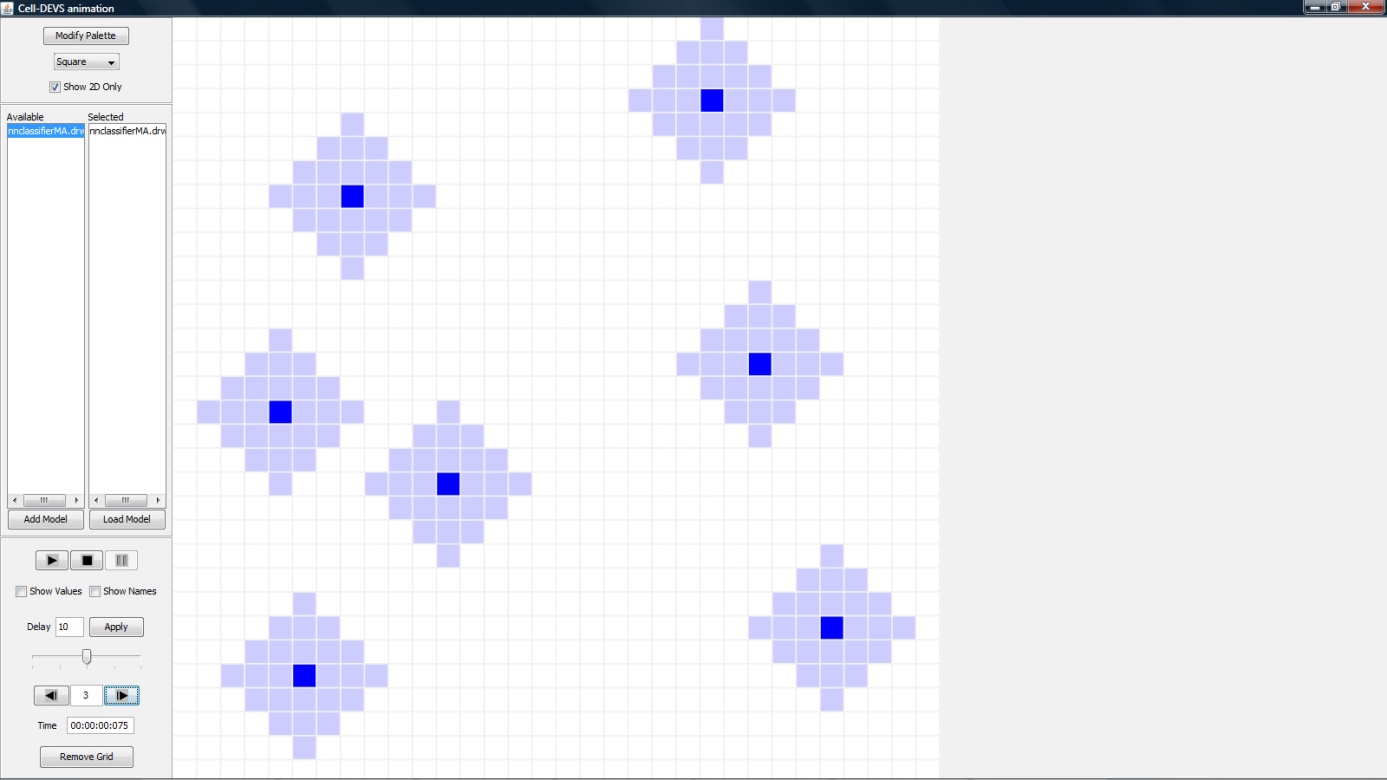
NOTE: It should be noted that since we are using a non-wrapped space, the boundaries of our feature space will be assigned boundary values, which is correct.

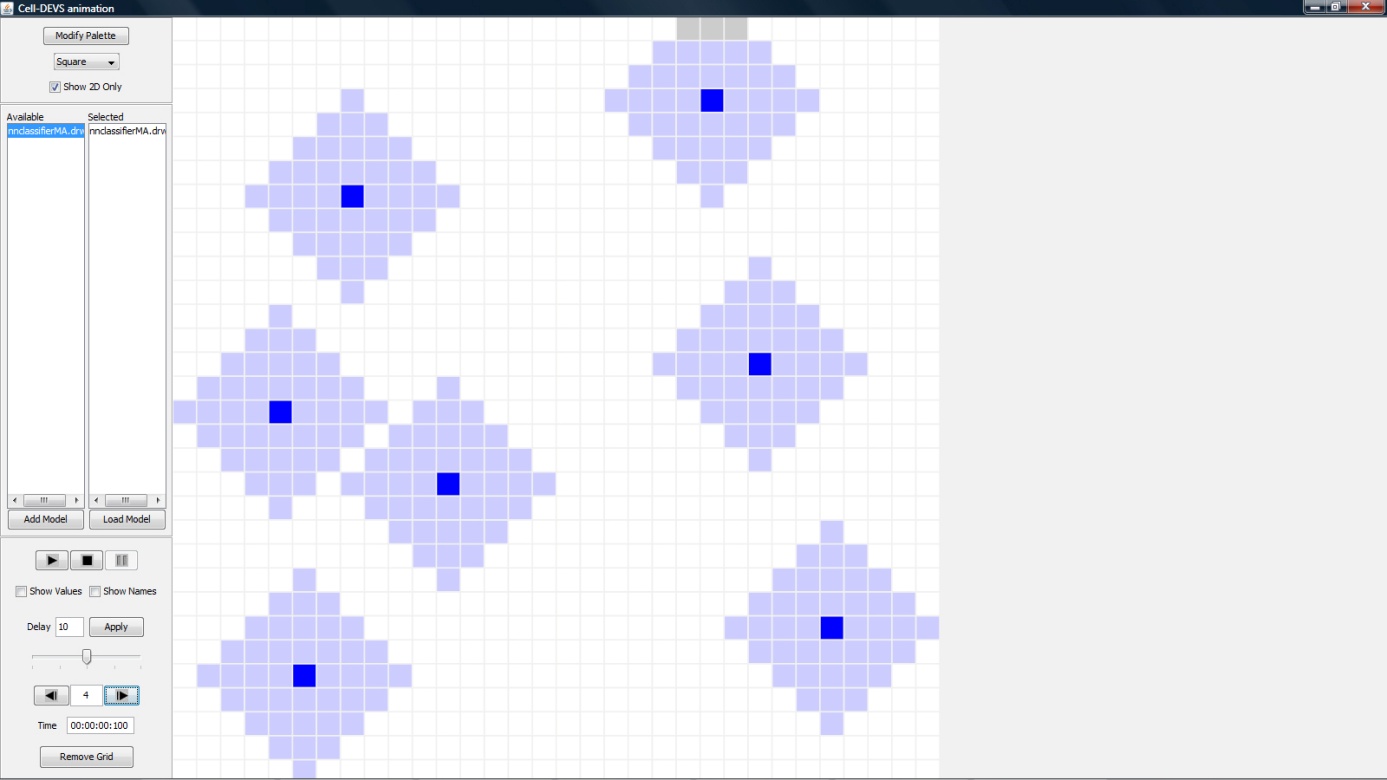
The result of running the model is provided below:

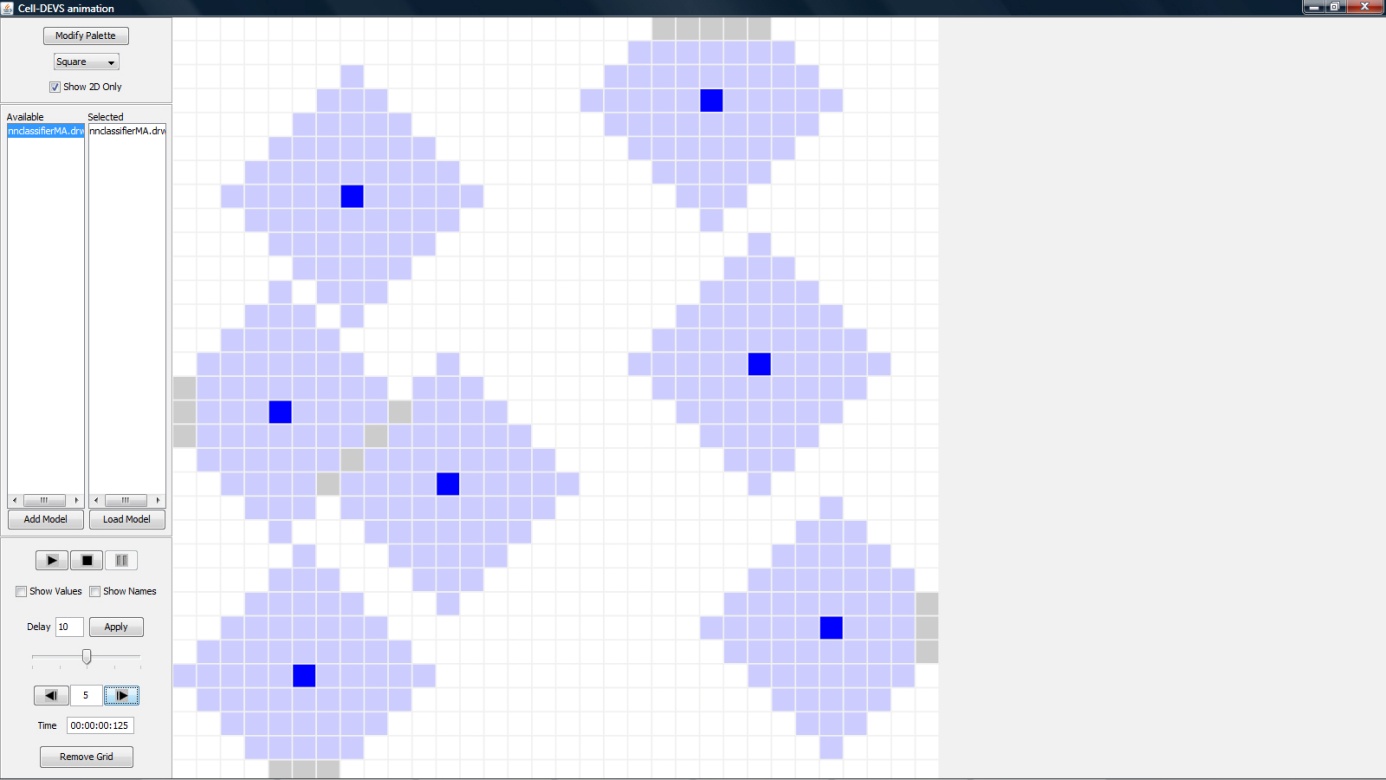


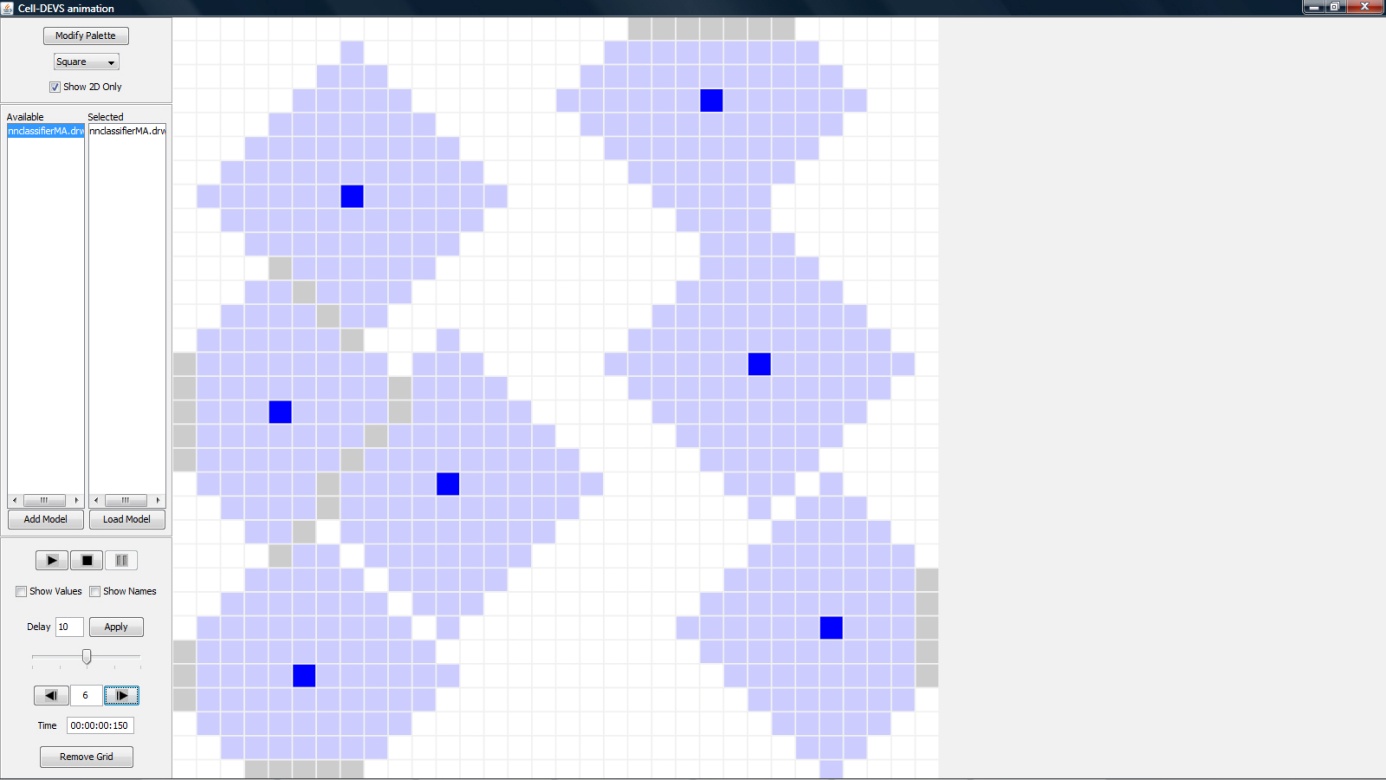


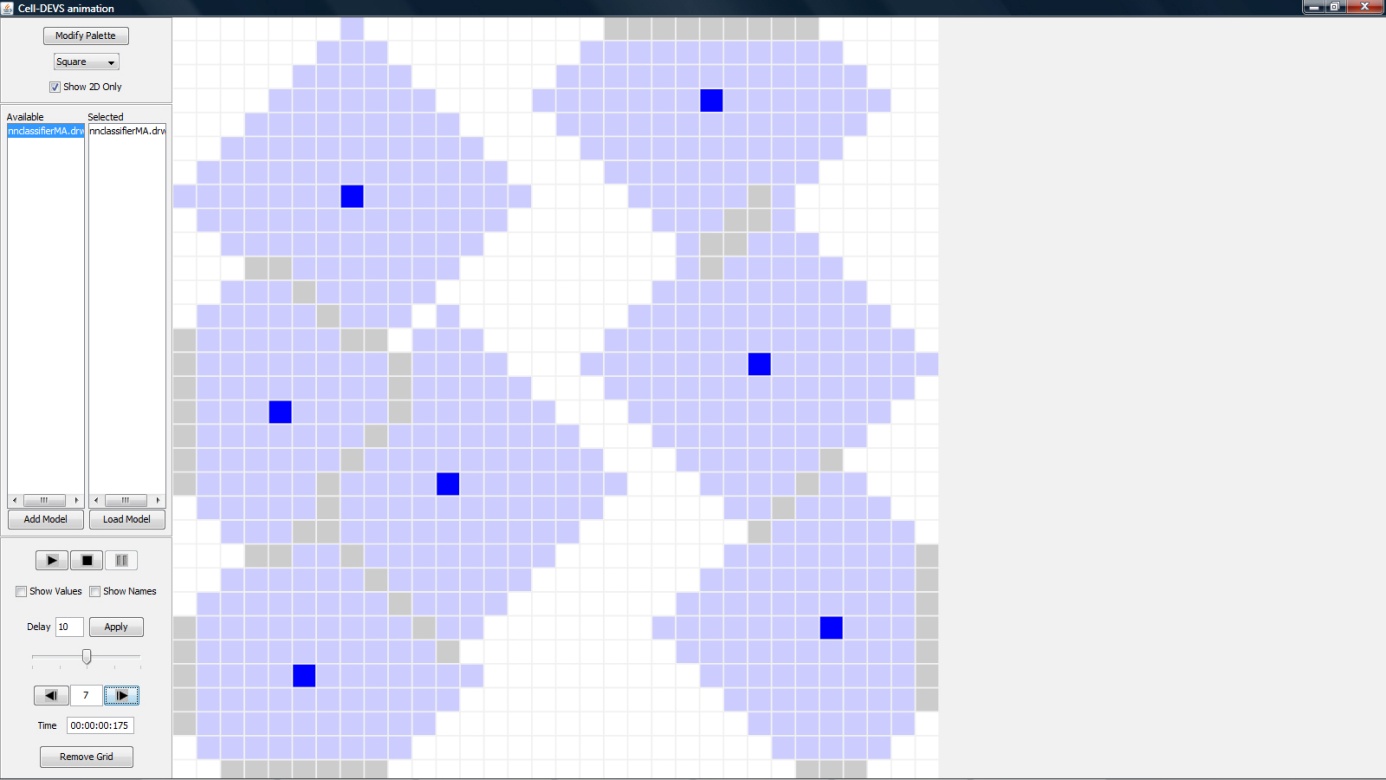


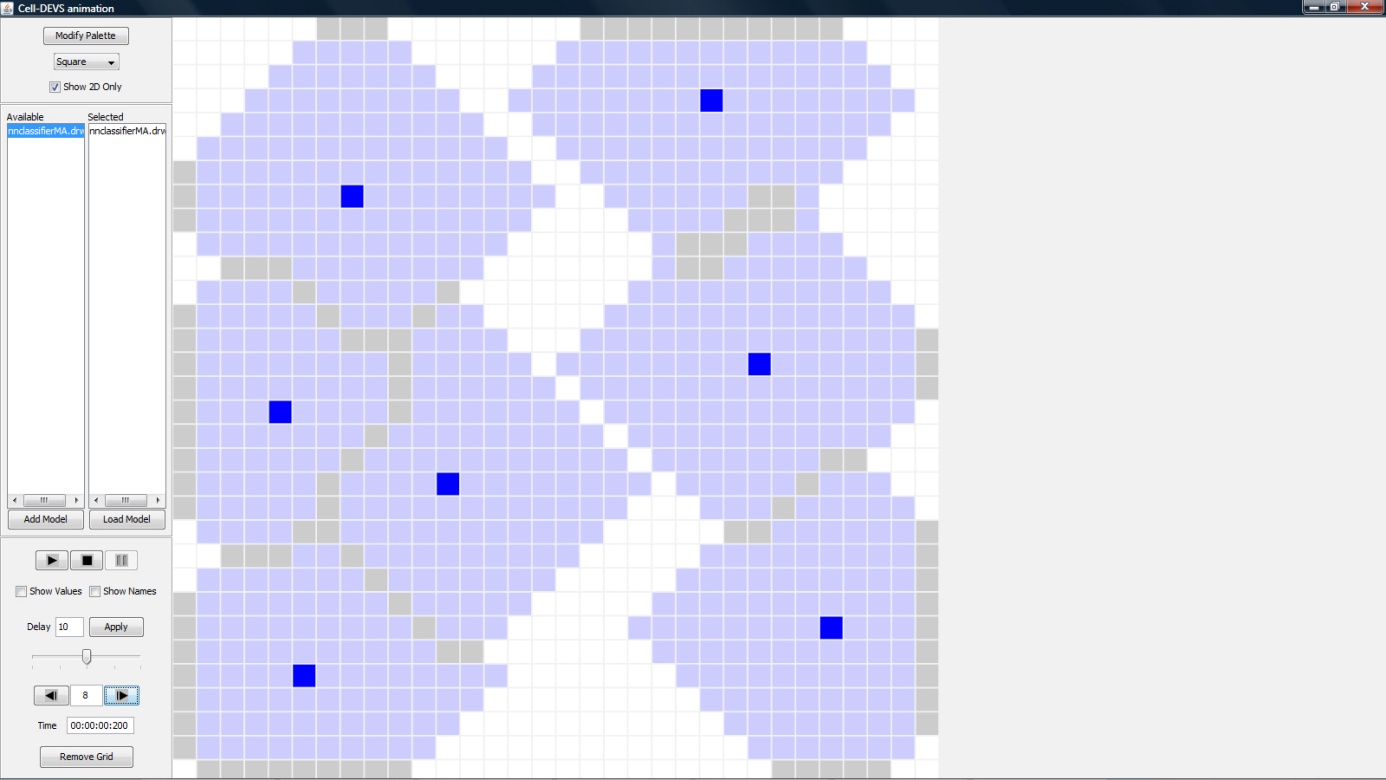


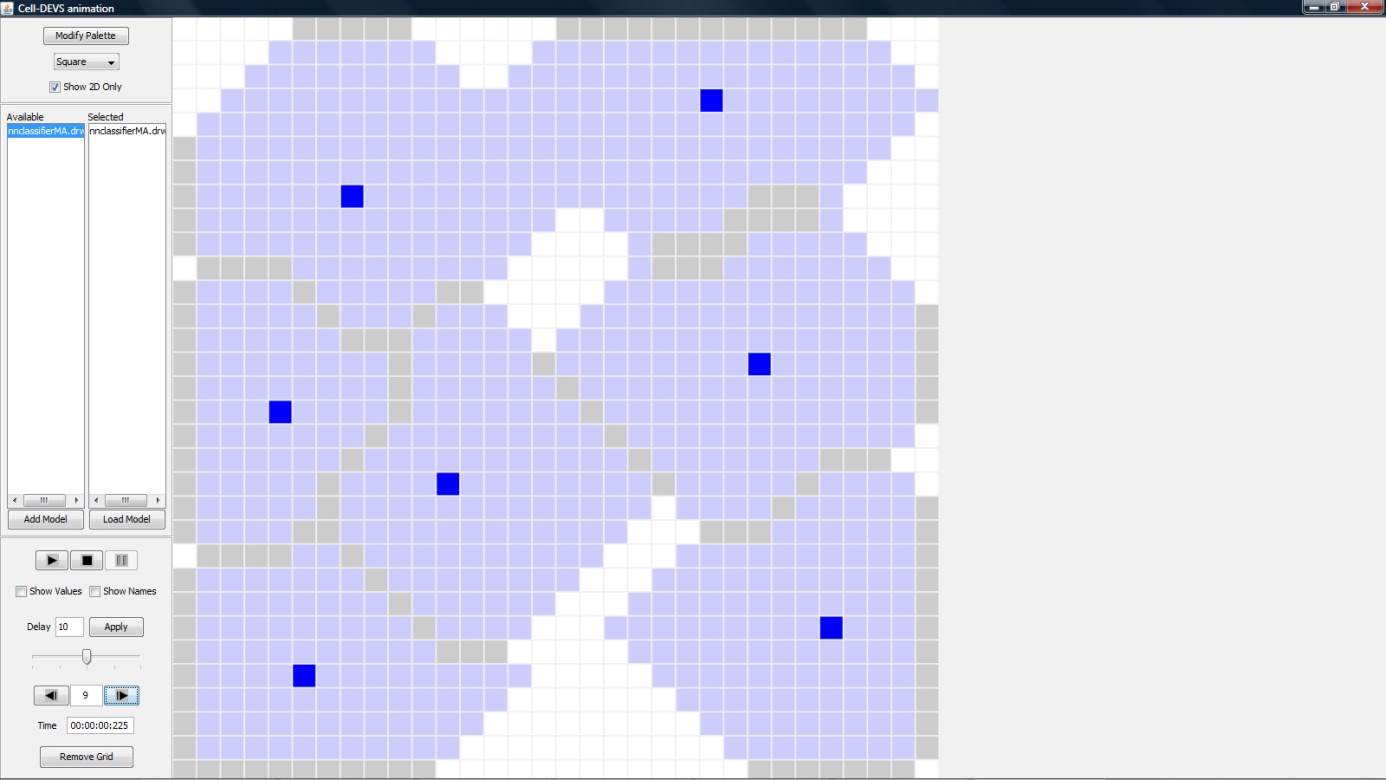


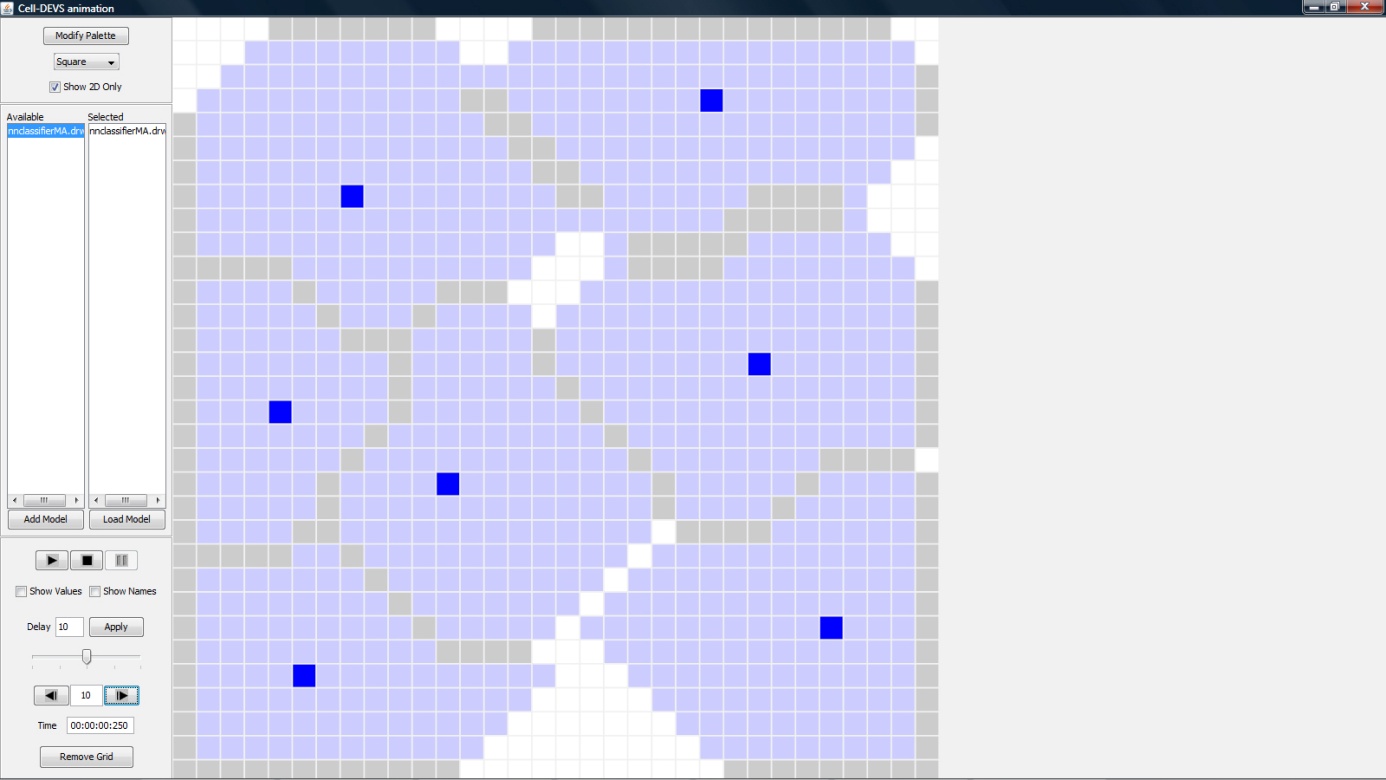


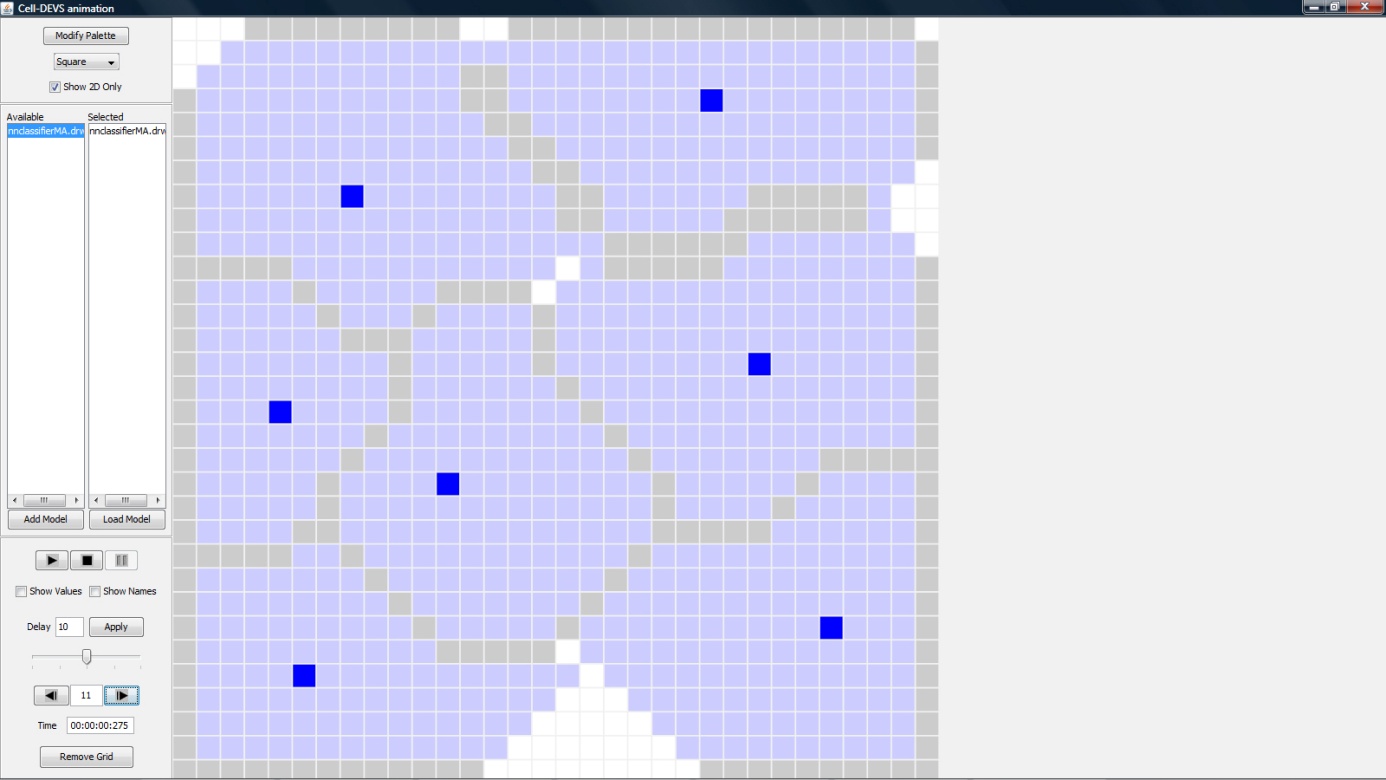


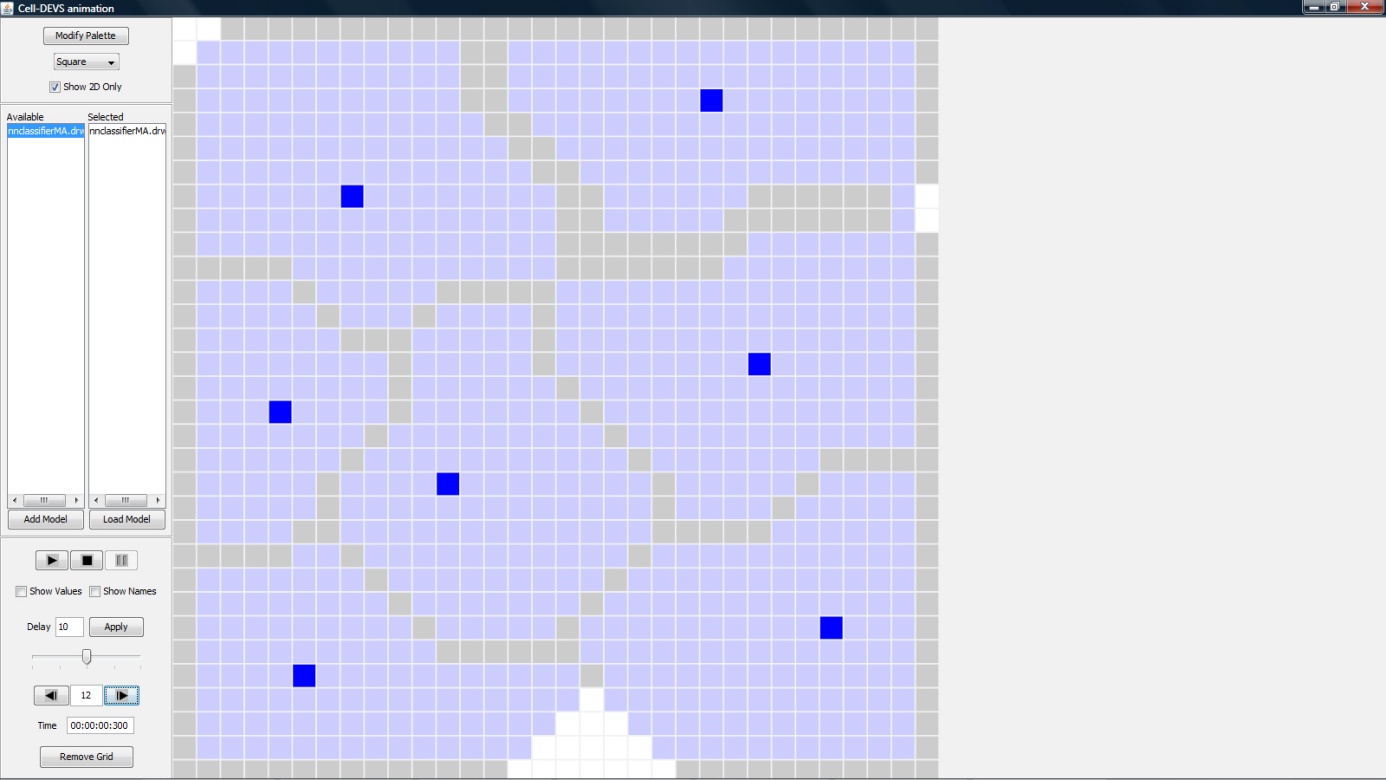


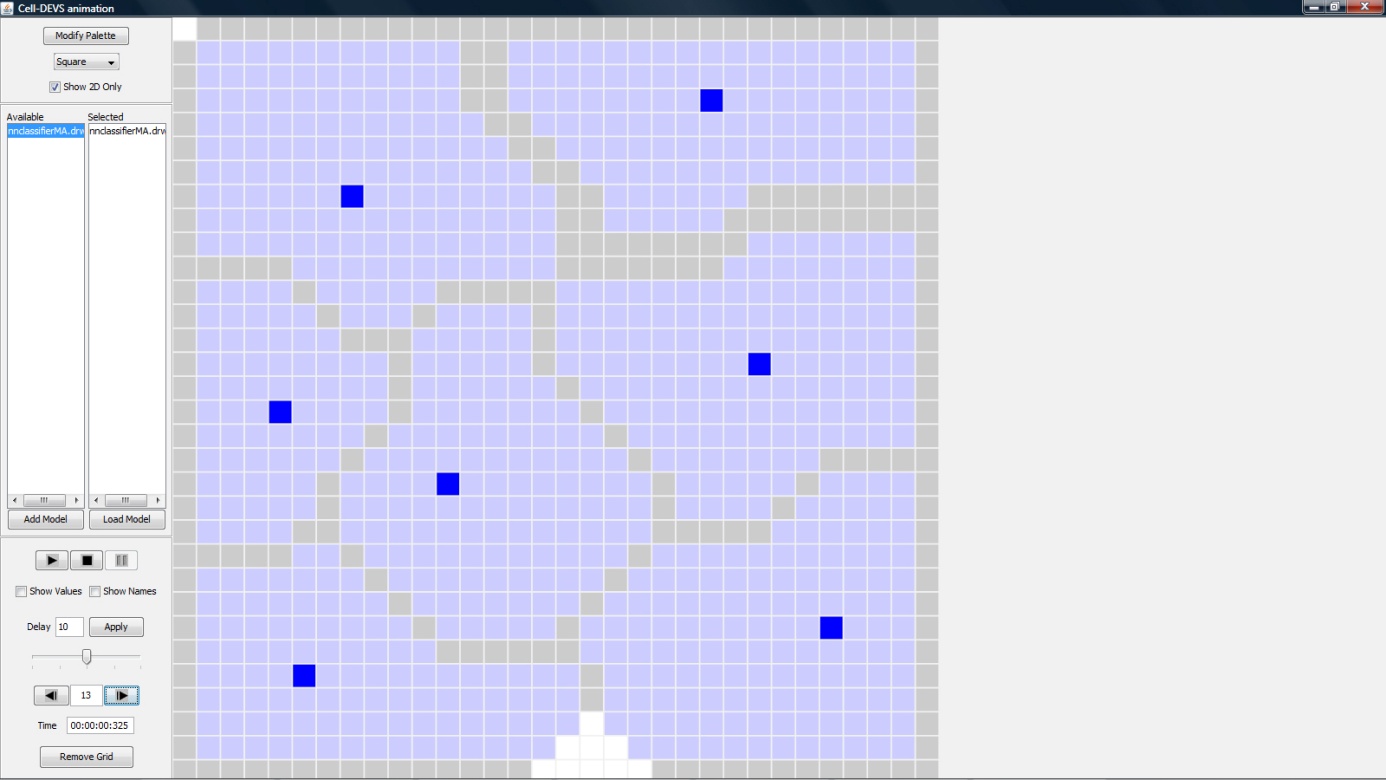


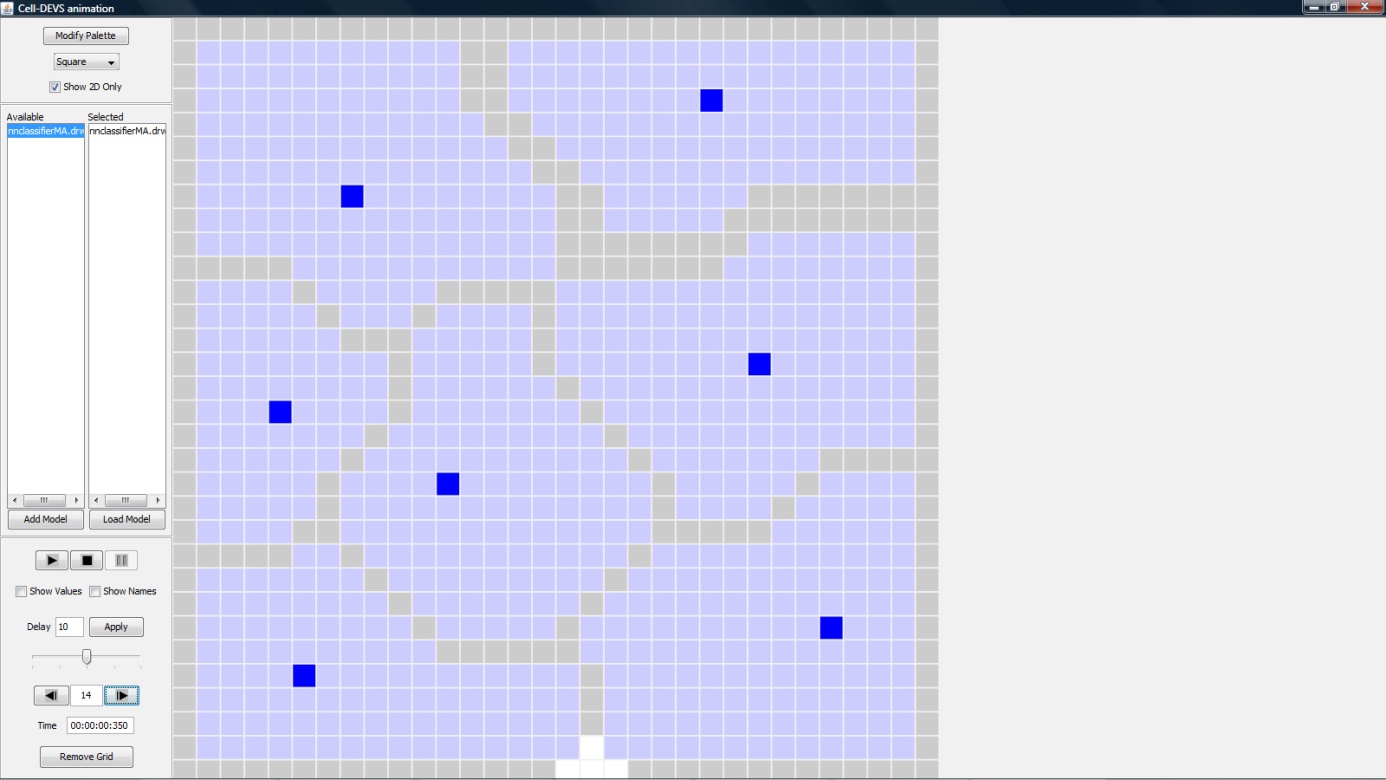


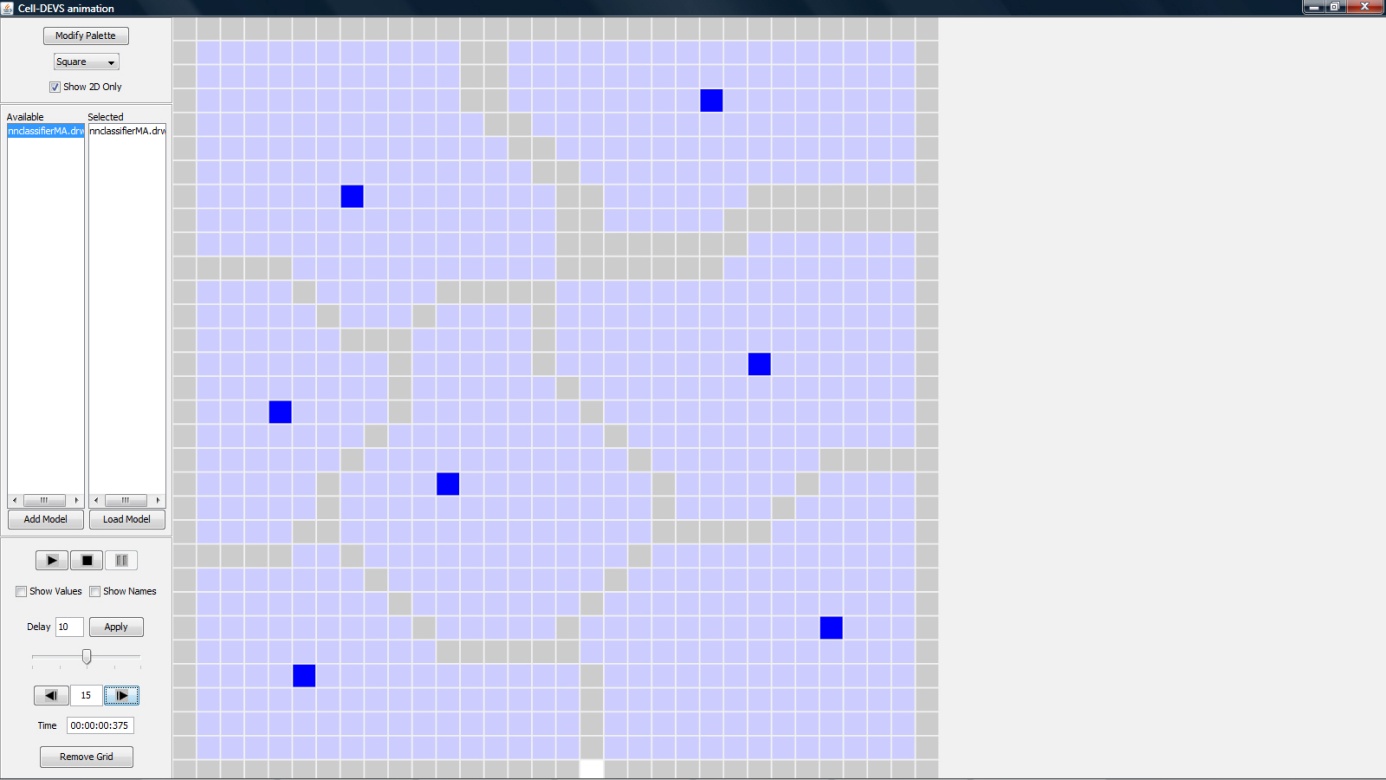


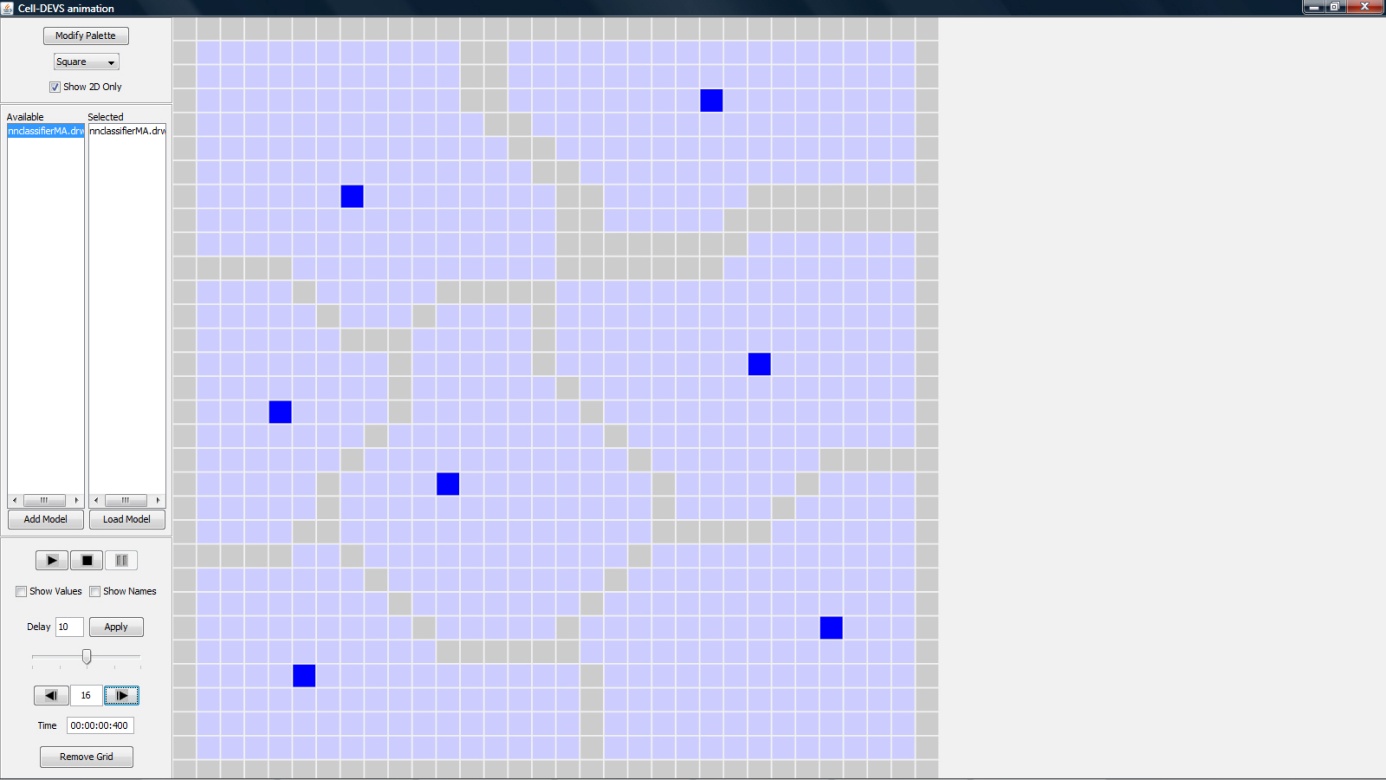






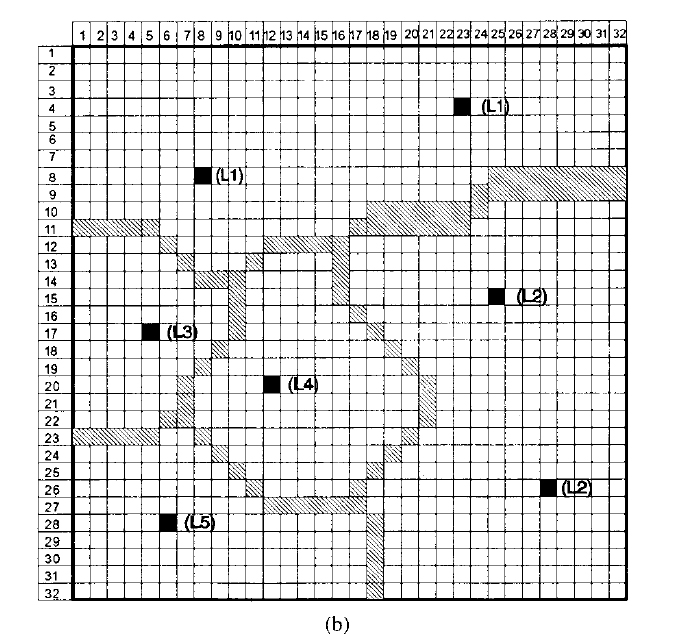






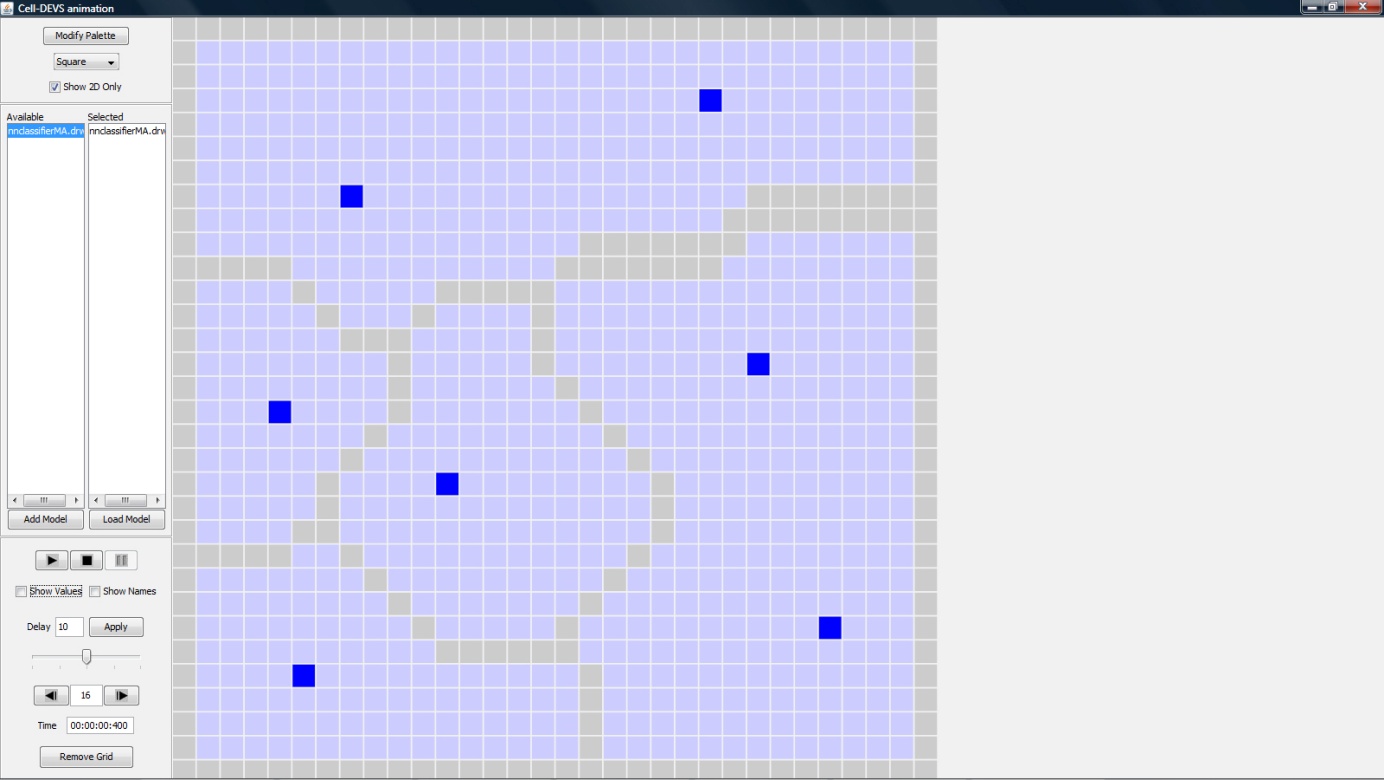
This outcome is exactly similar to that of [1].

In [1], the authors test the classifier by equalizing two pair of samples. The outcome of [1] is presented in Figure 6.

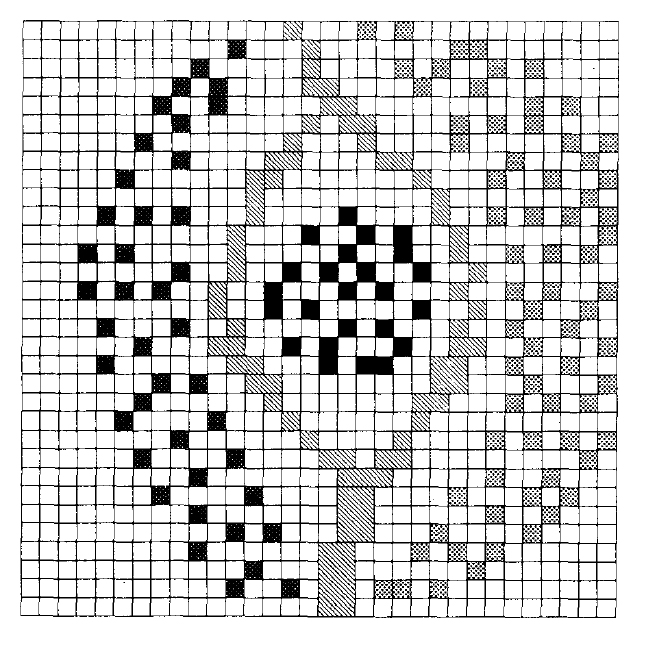


**Fig 5. Directly reproduced from [1]**

This is modelled by “nn2.val”. The outcome of our model is shown below which is identical to Figure 5:

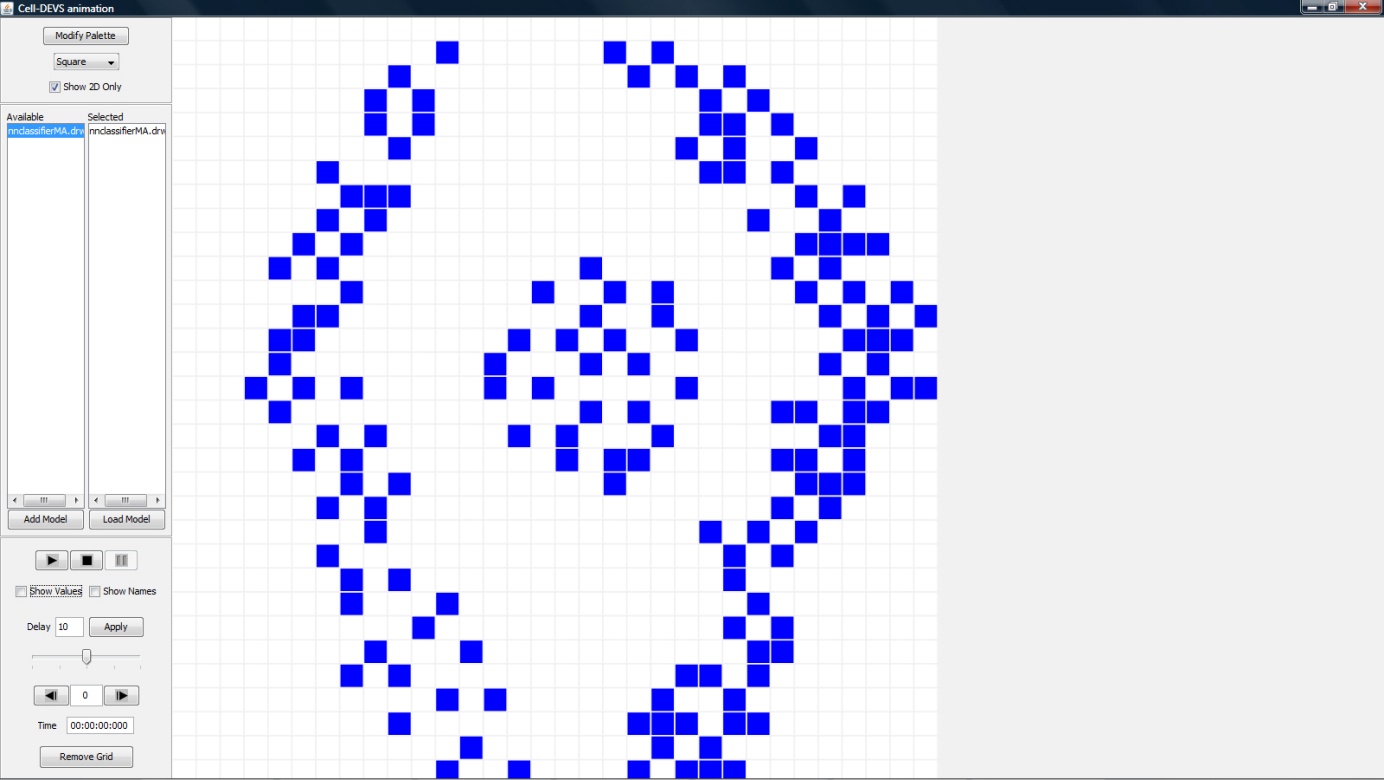


A variety of different test sets have been employed in [1]. Figure 6 illustrates one of the important sets which is also used for testing this model.

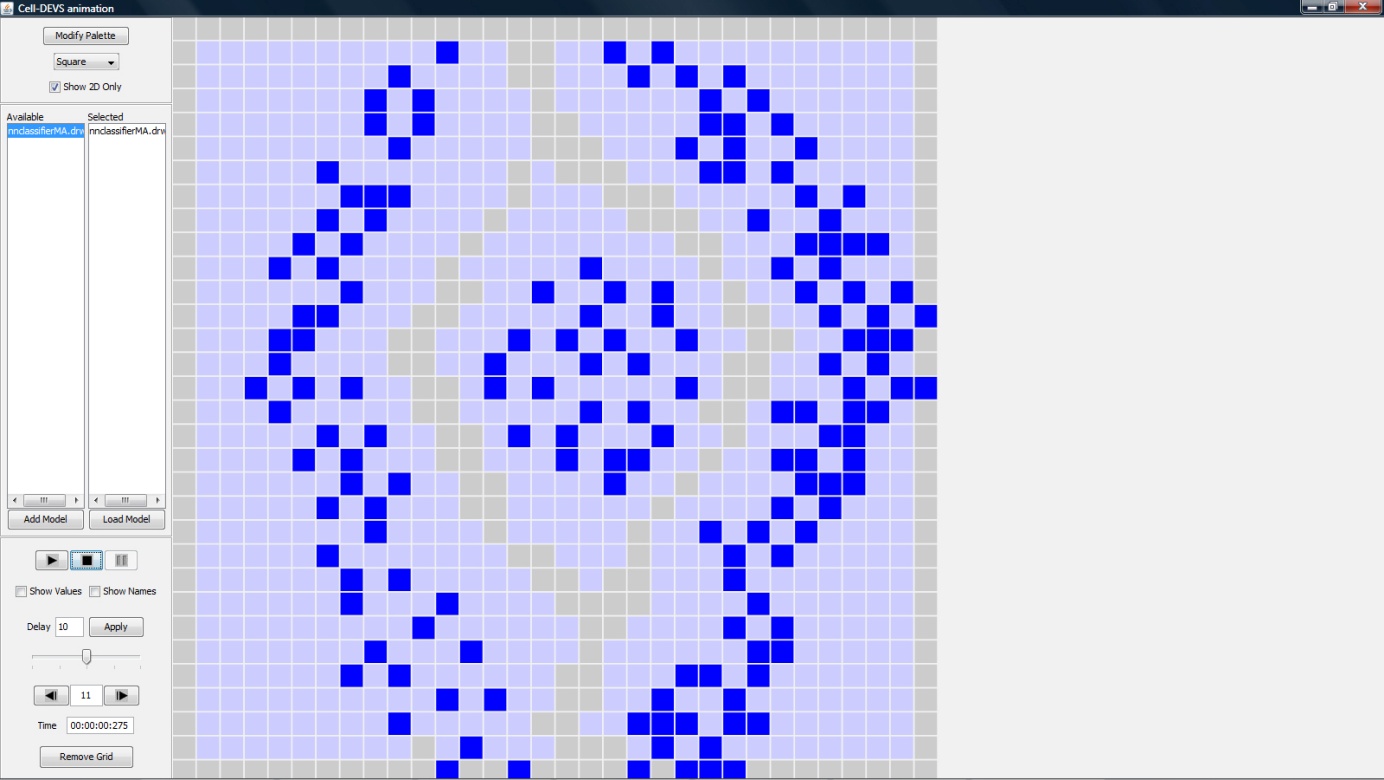


**Fig 5. Directly reproduced from [1]**

We have tried to mimic this set by “nn3.val” as the following:



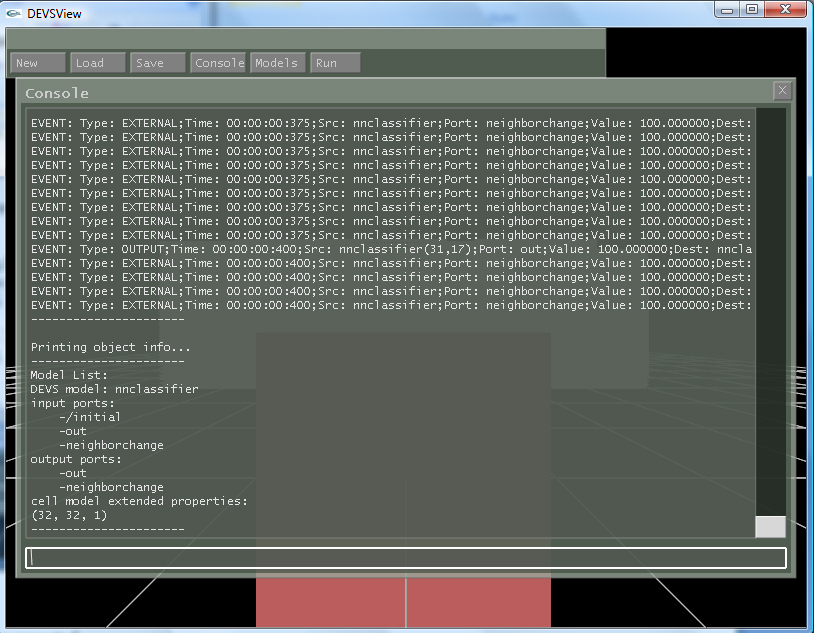
The following image is the output after the simulation is complete:



**\*\*\* A movie of each of the three runs is also available in the folder.**

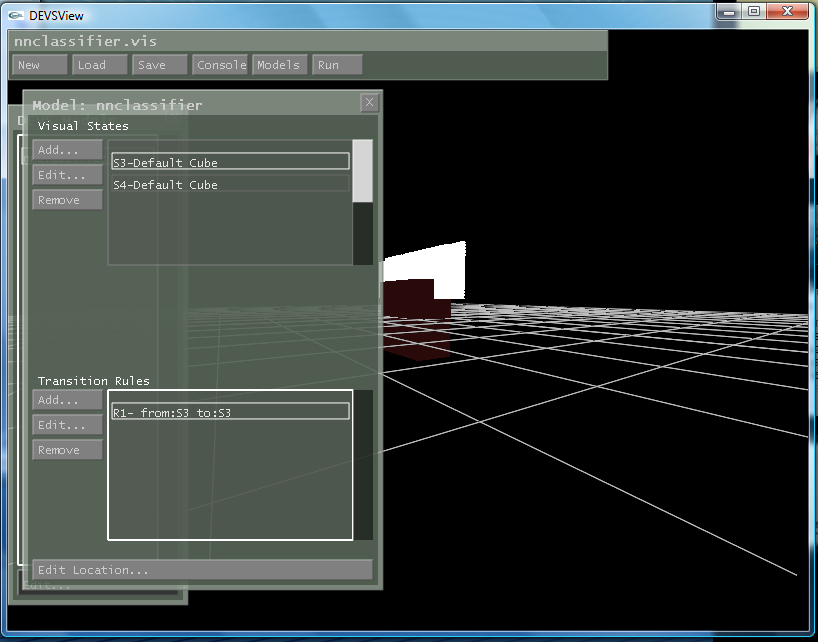
**Bonus part:**

The model was imported to DEVSView. The image below shows the initial step of the process:



The model list is as expected.

The next step is shown below:



From here on however, I could not figure out how to run the model. The partially complete “nnclassifier.vis” is included in the folder.

**References**

[1] P. G. Tzionas, P. G. Tsalides, and A. Thanailakis, “A new, cellular automaton-based, nearest neighbour pattern classifier and its VLSI implementation”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 2,  [Issue 3](http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=7556&isYear=1994), Sep 1994, pp. 343 – 353.

[2] R. O. Duda and P. E. Hart, Pattern ClassiJication and Scene Analysis. New York: Wiley, 1973.