Computing selected topics

- Prácticas segundo parcial -

Grupo 3CM8

Vargas Romero Erick Efraín
Prof. Juárez Martínez Genaro

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Juan de Dios Bátiz, nueva industrial Vallejo
07738 ciudad de México

Contents

Chapter 1

Autómata celular

1.1 Introducción

Los autómatas celulares son modelos matemáticos que valga la redundancia, modelan sistemas dinámicos, los cuales evolucionan con el paso del tiempo. Los autómatas celulares fueron descubiertos por John von Neumann en la década de 1940, y fue descrito en su libro Theory of Self-reproducing Automata. John von Neumann tenía como objetivo modelar una máquina, que fuese capaz de auto replicarse, al intentar esto llego a un modelo matemático, el cual describe a dicha máquina con ciertas reglas sobre una red rectangular.

1.2 Definición

Como ya se ha mencionado anteriormente, un autómata celular es un modelo matemático para un sistema dinámico, este sistema evoluciona con el paso del tiempo. El autómata celular está compuesto por células o celdas las cuales adquieren ciertos valores o estados. Al ser este un sistema dinámico y al evolucionar a través del tiempo los estados o valores que tienen las células cambian de un instante a otro, esto en unidades de tiempo discreto, en otras palabras es posible hacer una cuantización. Siendo así, el cunjunto de células evolucionan según la expresión matemática, la cual evolucionará según los estados de las células vecinas, a esto se le conoce como regla de transición local.

1.3 Componentes

1.3.1 Un espacio rectangular

El autómata celular está definido ya sea en un espacio de dos dimensiones o bien en un espacio de n dimensiones, este es el espacio de evoluciones y cada una de las divisiones de este espacio es llamada célula.

1.3.2 Conjunto de estados

Los estados son finitos y cada elemento de la célula tomará un valor de este conjunto de estados. A cada vecindad diferente le corresponde un elemento del conjunto de estados.

1.3.3 Vecindades

Conjunto de contiguo de células cuya posición es relativa respecto a cada una de ellas. Como se mencionó anteriormente, a cada vecindad diferente le corresponde un estado diferente del conjunto de estados.

1.3.4 Función local

Es la regla de evolución que determina el comportamiento del autómata celular. Esta regla esta conformada por una célula central y sus vecindades. También esta define como debe cambiar de estado cada una de las células dependiendo de los estados de las vecindades anteriores. Esta función puede ser representada como una función algebraica o como un conjunto de ecuaciones.

1.4 Límites o fronteras

Podemos hacer una representación visual de los autómatas celulares, y para que podamos entenderlo de mejor manera es necesario mencionar los límites y las fronteras, del espacio en el cual existe el autómata celular.

1.4.1 Frontera abierta

Considera que todas las células fuera del espacio del autómata tienen un valor el cual es fijo.

1.4.2 Frontera reflectora

Las células fuera del espacio del autómata toman los valores que están dentro como si se tratase de un espejo.

1.4.3 Frontera periódica o circular

Las células que están en los límites o en la frontera interaccionan con sus vecinos inmediatos y con las células que están en el extremo opuesto del arreglo, como si el plano estuviese doblado a manera de cilindro.

1.4.4 Sin frontera

La representación de autómatas no tiene límite alguno, es infinito.

1.5 Práctica

1.5.1 Descripción

Para esta práctica se ha creado nuestro primer autómata celular, el cual cumple la función local del "Juego de la vida". El juego de la vida es un autómata celular que fue diseñado por el matemático John Horton Conway en 1970. Este se trata de un juego de cero jugadores, es decir, el estado de evolución está definido por el estado inicial y no requiere entrada de datos alguna posteriormente. El tablero de este juego es una matriz formada con células (espacios cuadrados) que se extienden por el infinito a toda dirección. Cada célula tiene ocho células vecinas, que son las que están más próximas a ella, incluidas las diagonales, de forma gráfica

	Célula central	

Las reglas para ese juego son las siguientes

- Una célula muerta con exactamente 3 células vivas "nace" (En la siguiente generación estará viva)
- Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso esta muere.

Pero para esta práctica no solo es posible utilizar estas reglas, si es necesario podemos cambiar estas reglas, donde los valores pueden ir desde 1 hasta el 8.
Para esta práctica se decidió por utilizar el lenguaje de programación C# y el entorno de desarrollo integrado Visual Studio 2015.

También en este programa se ha añadido código que permite generar todas las combinaciones posibles en submatrices desde 2x2 hasta 7x7 en el caso ideal, pero por cuestiones de memoria, solo se han generado como máximo 5x5, ya que el crecimiento en memoria es exponencial. Además se ha intentado el crear un algoritmo que realiza un filtrado en todos los grafos que son generados por estos fragmentos de código, pero al tratar de generar combinaciones muy grandes ciertas funciones del lenguaje no funcionan de manera correcta, ya que se realizan comparaciones un tanto extrañas.

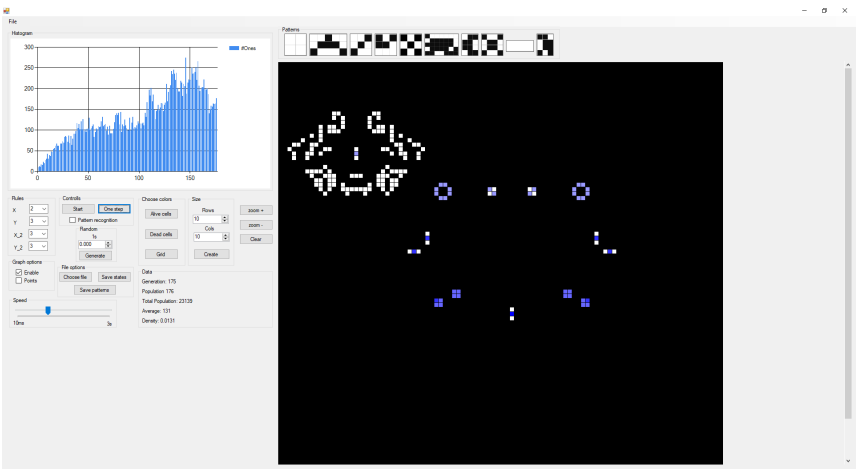
Es necesario añadir que ahora es posible observar cuando las células se quedan "estancadas" es decir cada ciertas generaciones se puede observar que se actualiza el color de las células que se han quedado en el estado de vivas por mucho tiempo.

Finalmente, se ha añadido un algoritmo de reconocimiento de patrones el cual se ejecuta simultáneamente con el programa del juego de la vida de John Conway, el algoritmo es bastante simple y podríamos decir que eficiente ya que aprovecha la información que fue generada por el algoritmo descrito en el párrafo anterior, podríamos decir que es similar a programación dinámica.

1.5.2 Pruebas

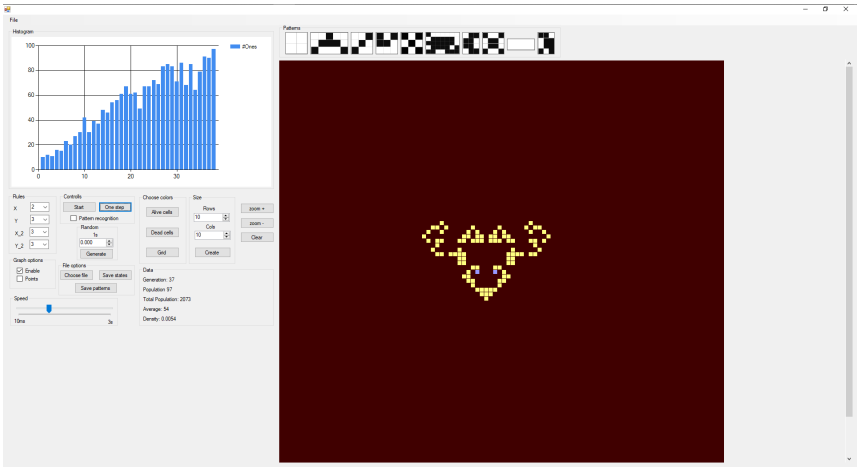
Este programa tiene varios aspectos que han sido cubiertos. Para iniciar las pruebas se han añadido algunas células vivas haciendo click en el elemento que inicialmente es color negro. Esto se realiza haciendo uso de algunos eventos, tal y como se muestra en la siguiente figura.

Figure 1.1: Prueba haciendo uso de eventos



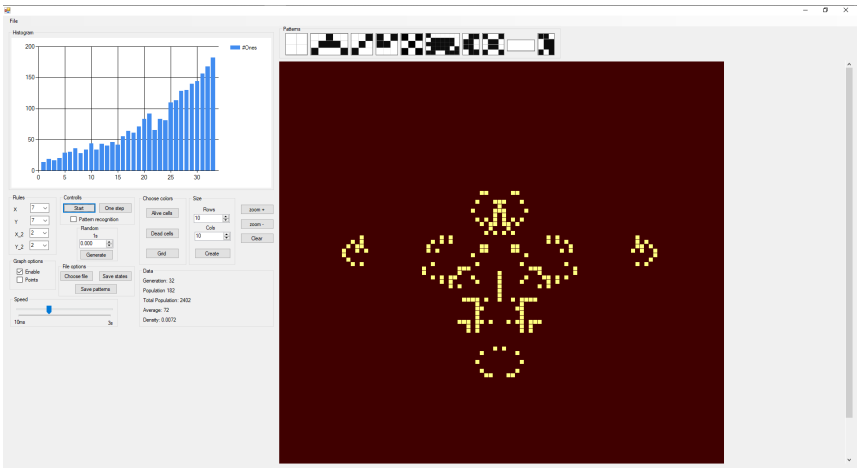
A continuación se muestra el uso del cambio de colores. Podemos seleccionar cualquiera de los colores RGB disponibles, estos podemos aplicarlos tanto al color de células muertas, como células vivas y el grid que tiene la matriz

Figure 1.2: Uso de colores



Las dos pruebas anteriores han sido realizadas colocando las reglas de "Game of life", estas reglas son colocadas por defecto, si no hay establecido algún parámetro. Para la siguiente prueba se han realizado cambios por las reglas de "Diffusion"

Figure 1.3: Prueba regla de difusión



Otro de los parámetros cubiertos es el guardar alguna generación que queramos, por ejemplo la última generada con las reglas de difusión.

Figure 1.4: Guardando datos

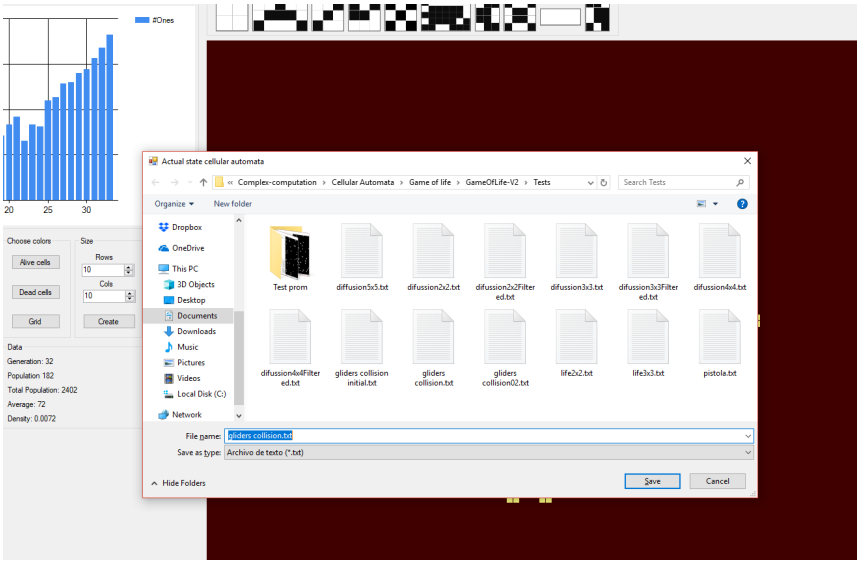
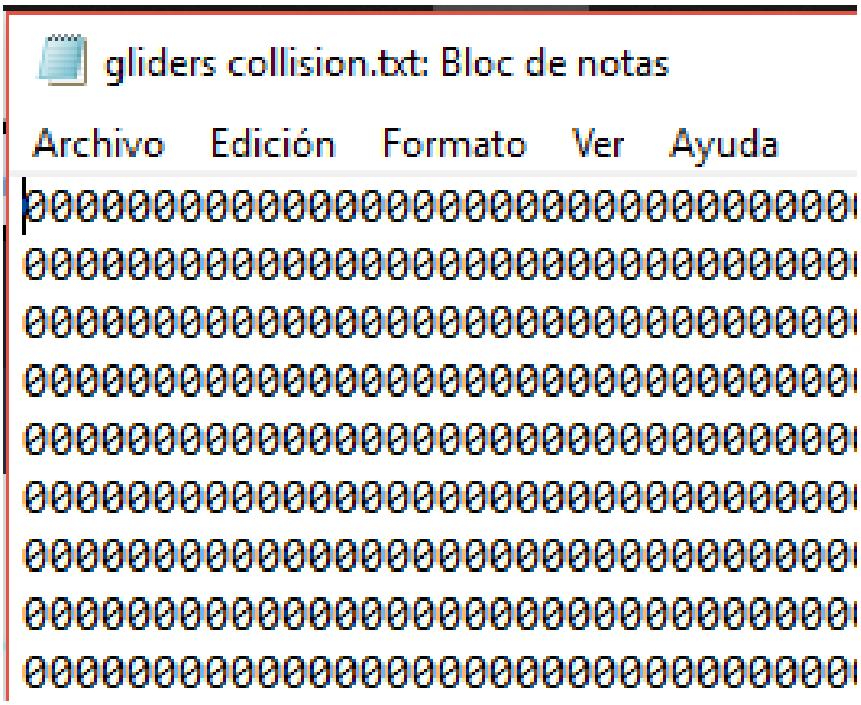
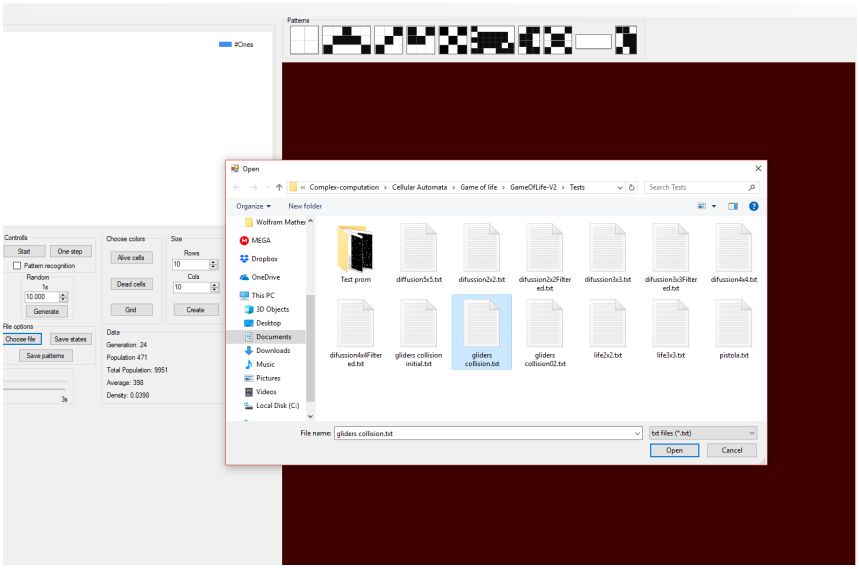


Figure 1.5: Contenido del archivo



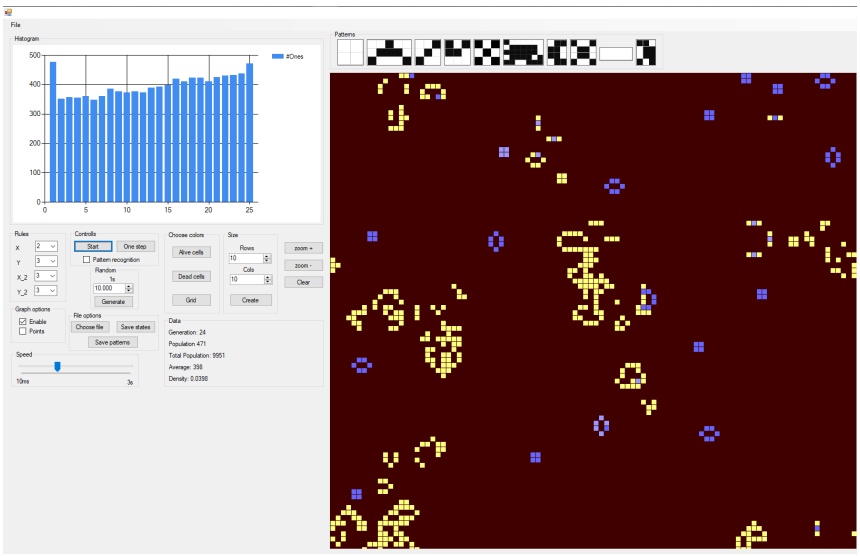
También como era de esperarse algún archivo de texto también puede ser cargado al programa.

Figure 1.6: Recuperando datos



Finalmente es posible generar un random para llenar matriz. El usuario tiene la posibilidad de elegir la probabilidad con la que desea que aparezcan los unos. Para esta actualización se le da la opción al usuario de ingresar incluso probabilidades con punto decimal.

Figure 1.7: Random



1.6 Generador de patrones

También el programa dispone de una sección la cual permite generar un archivo el cual contiene todas las combinaciones posibles en submatrices de un tamaño n (idealmente) pero al tener una cantidad enorme de combinaciones y por cuestiones de memoria solo se han realizado los cálculos de las combinaciones existentes dentro de matrices de hasta cinco por cinco elementos. Al utilizar esta opción se crean dos archivos, el primero contiene todas las combinaciones existentes en la submatriz de dimensión n que ha ingresado el usuario, para este caso fue de tres, además de generar otro archivo el cual hace uso de un algoritmo que hace el intento por filtrar todos los grafos existentes en el archivo original, pero al ser un problema tipo no polinomial al aplicar esto con submatrices mayores a cuatro empieza a tener un comportamiento un tanto extraño debido a que la información utilizada para filtrar cada grafo es demasiada. Finalmente se han graficado estas figuras en Mathematica y se han generado los siguientes resultados

Figure 1.8: Grafos según la regla "Game of life" archivo original matriz 3x3

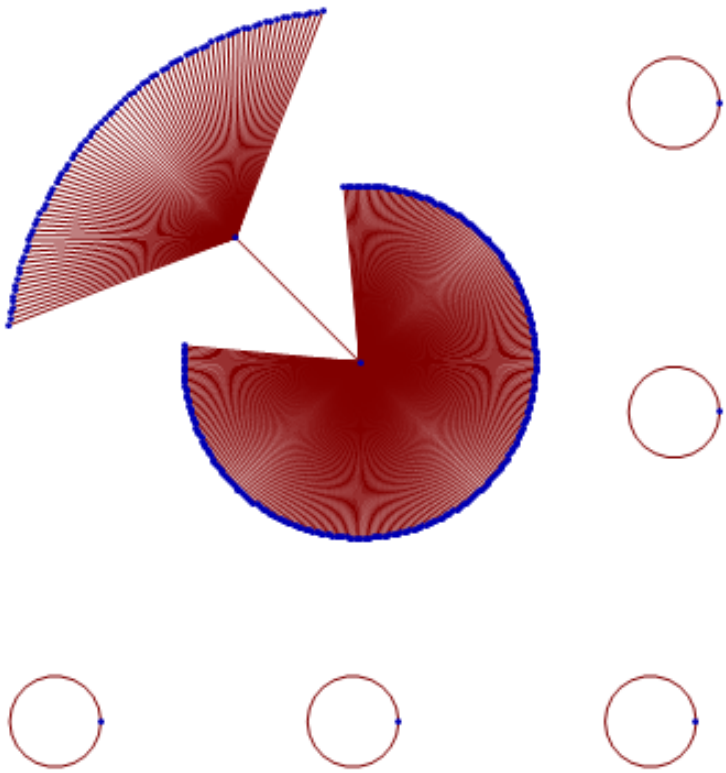
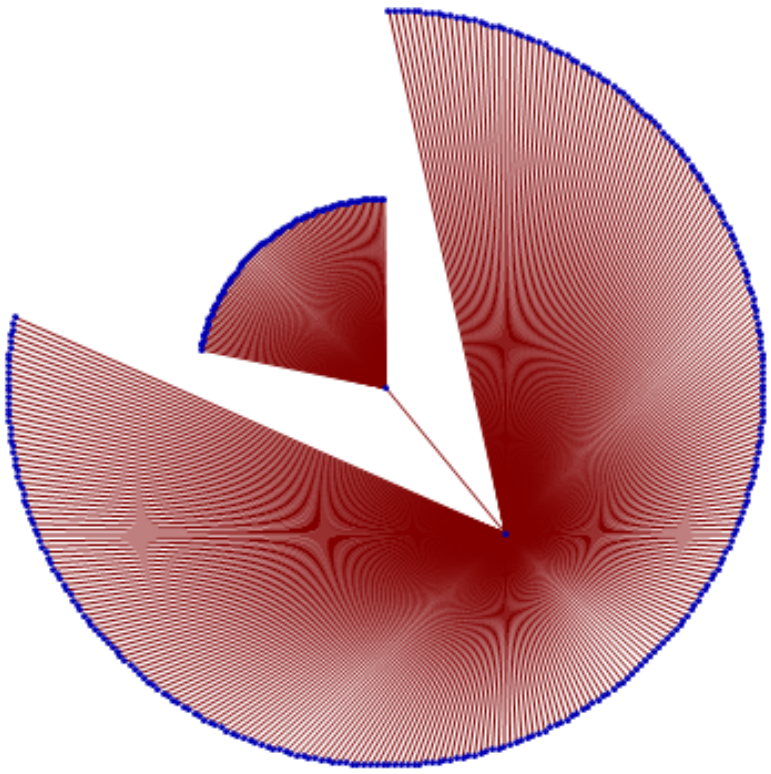
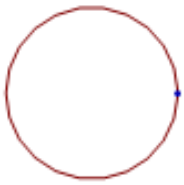


Figure 1.9: Grafos según la regla "Game of life" archivo filtrado matriz 3x3





A continuación se muestran los resultados obtenidos aplicando la misma regla en submatrices de dos por dos y cinco por cinco elementos, respectivamente

Figure 1.10: Grafos según la regla "Game of life" archivo original matriz 2x2

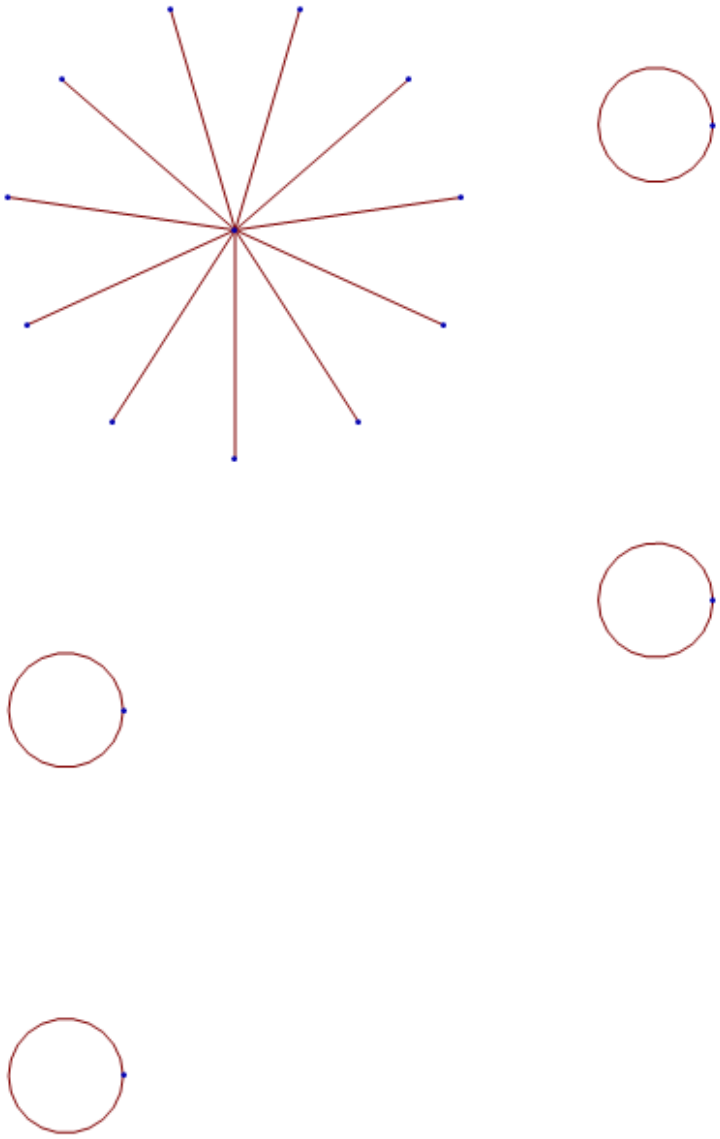
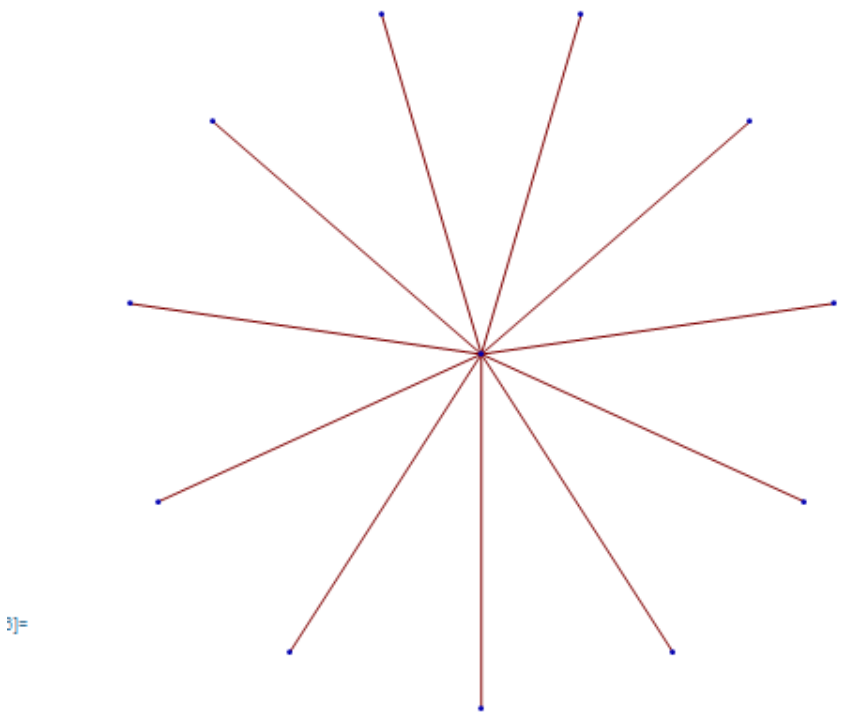


Figure 1.11: Grafos según la regla "Game of life" archivo filtrado matriz 2x2



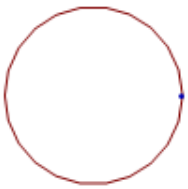


Figure 1.12: 01. Grafos según la regla "Game of life" archivo original matriz 4x4

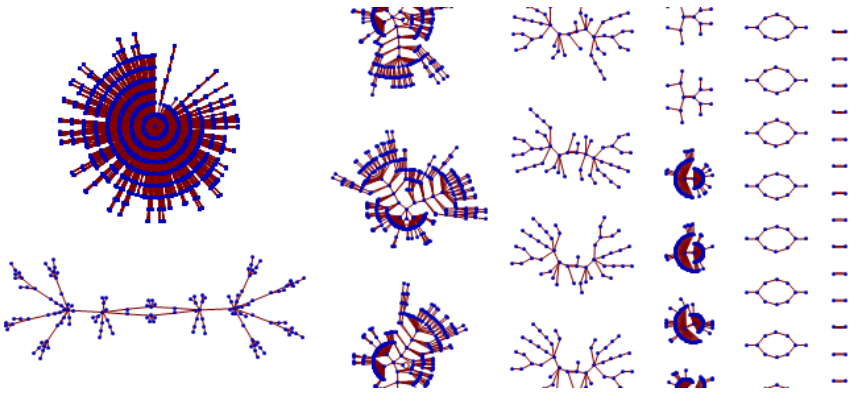


Figure 1.13: 02. Grafos según la regla "Game of life" archivo original matriz 4x4

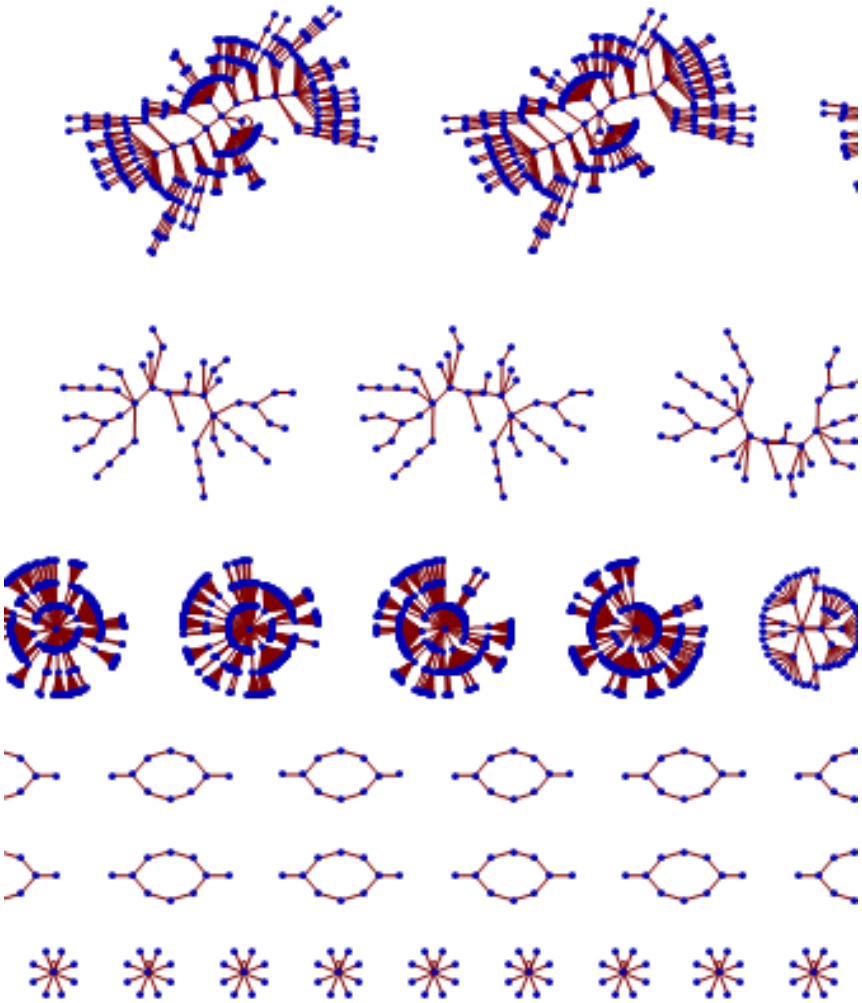
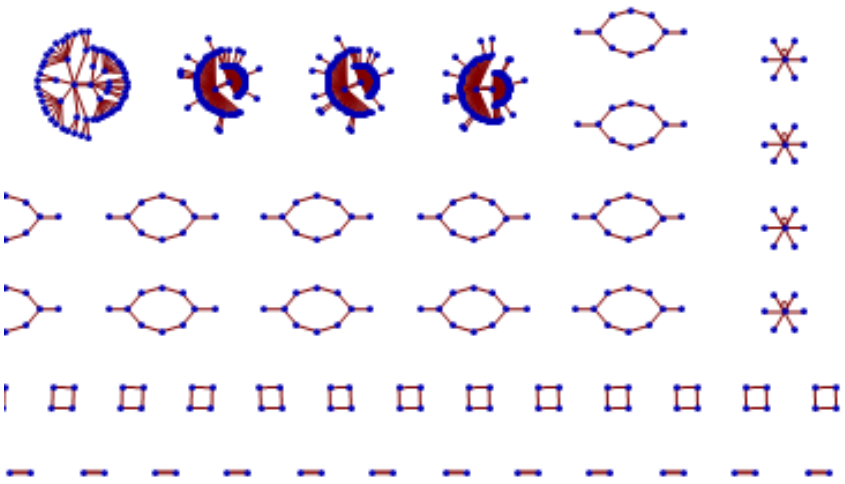
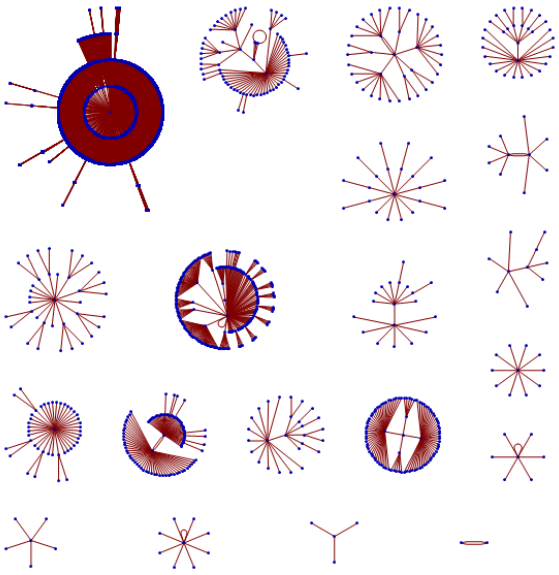


Figure 1.14: 03. Grafos según la regla "Game of life" archivo original matriz 4x4



Como podemos observar para estas figuras ya no se respetan las figuras originales o bien no todas, es verdad que se han generado grafos completamente diferentes pero no tienen gran similitud con el archivo original. Siendo así quizá el realizar una propia implementación de una tabla Hash podría ser una solución bastante óptima y eficiente para el generar estos grafos

Figure 1.15: Grafos según la regla "Game of life" archivo filtrado matriz 4x4



A continuación se realizó exactamente el mismo procedimiento que con la regla de

"Game of life" pero ahora aplicando la regla "Diffusion" y podemos observar que desde las submatrices más pequeñas se obtienen resultados considerablemente diferentes a la regla anterior. Se muestran los grafos iniciando con la submatriz de dos por dos elementos, posteriormente tres por tres y finalmente cuatro por cuatro elementos

Figure 1.16: Grafos según la regla "Diffusion" archivo original matriz 2x2

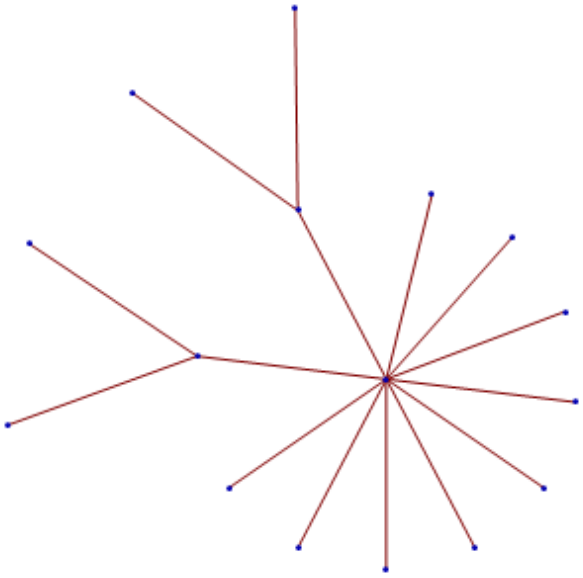


Figure 1.17: Grafos según la regla "Diffusion" archivo filtrado matriz 2x2

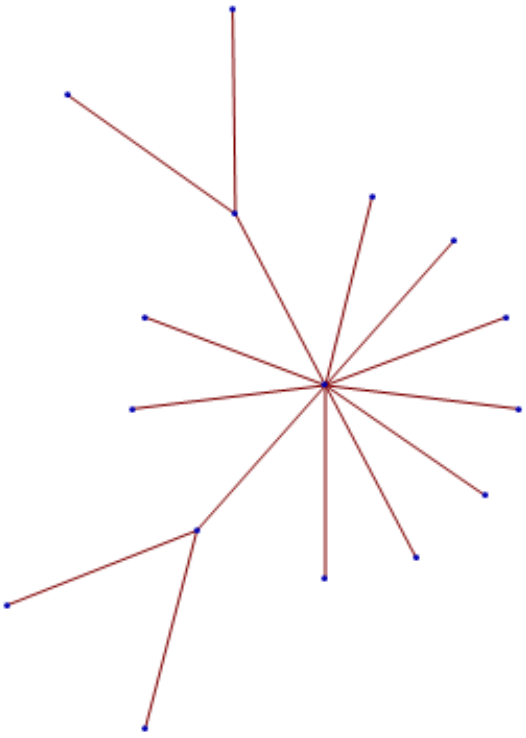


Figure 1.18: Grafos según la regla "Diffusion" archivo original matriz 3x3

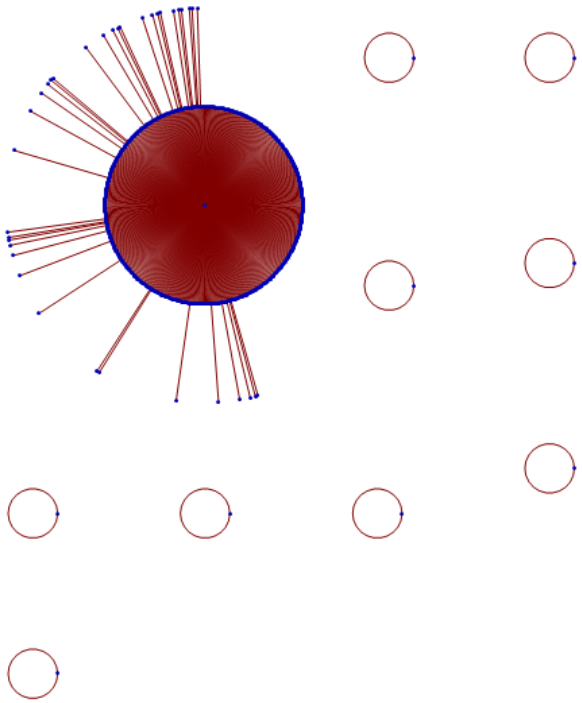


Figure 1.19: Grafos según la regla "Diffusion" archivo filtrado matriz 3x3

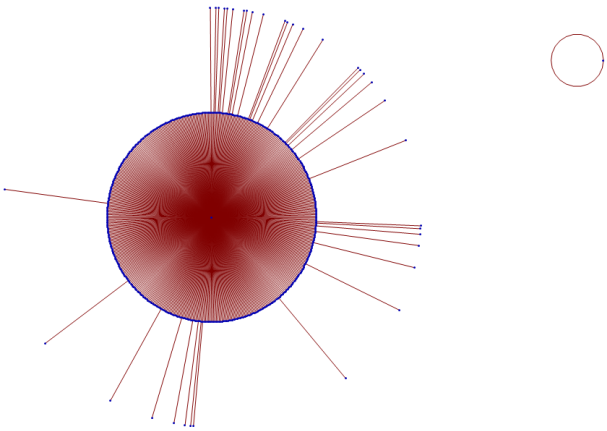


Figure 1.20: 01.Grafos según la regla "Diffusion" archivo original matriz 4x4

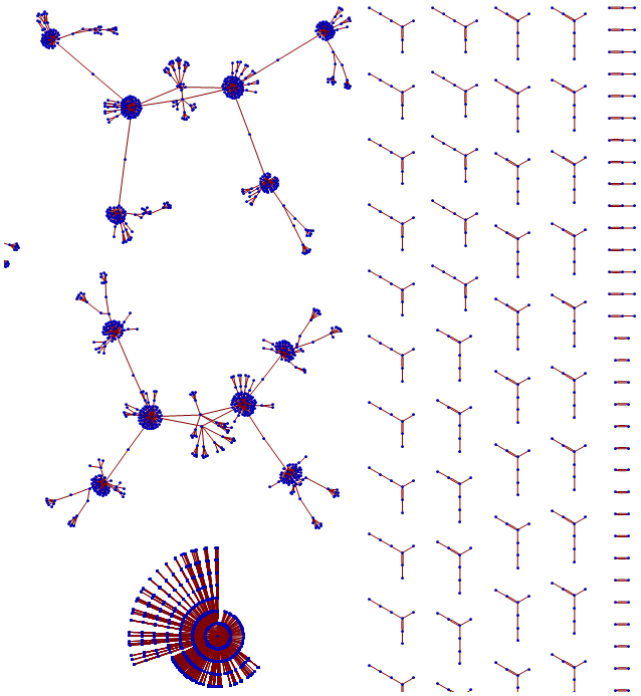


Figure 1.21: 02.Grafos según la regla "Diffusion" archivo original matriz 4x4

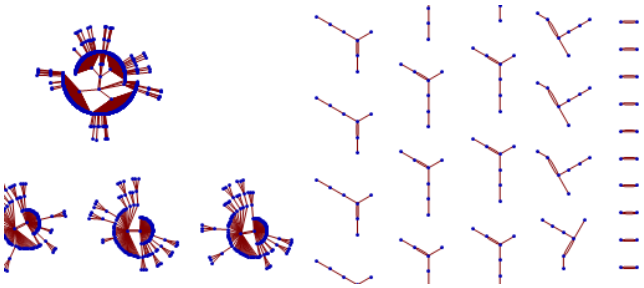
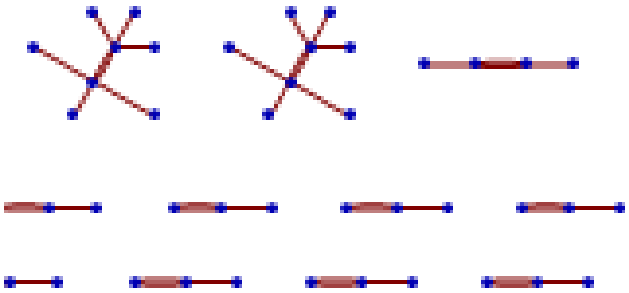
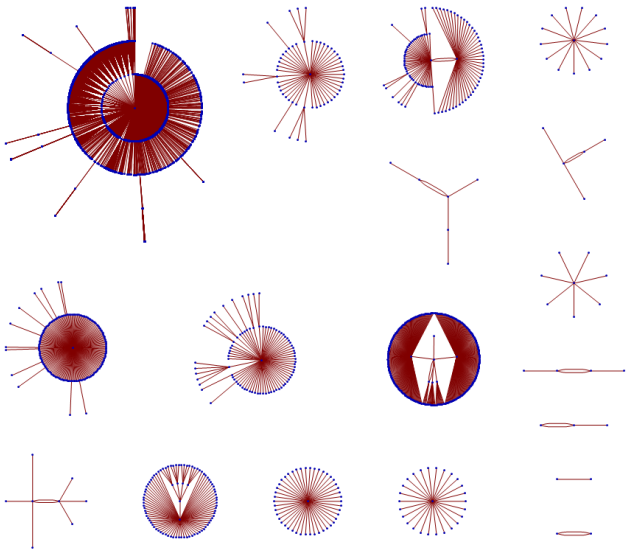


Figure 1.22: 03.Grafos según la regla "Diffusion" archivo original matriz 4x4



Nuevamente podemos notar que el filtrar los grafos cuando se tienen estructuras demasiado complejas no se hace correctamente.

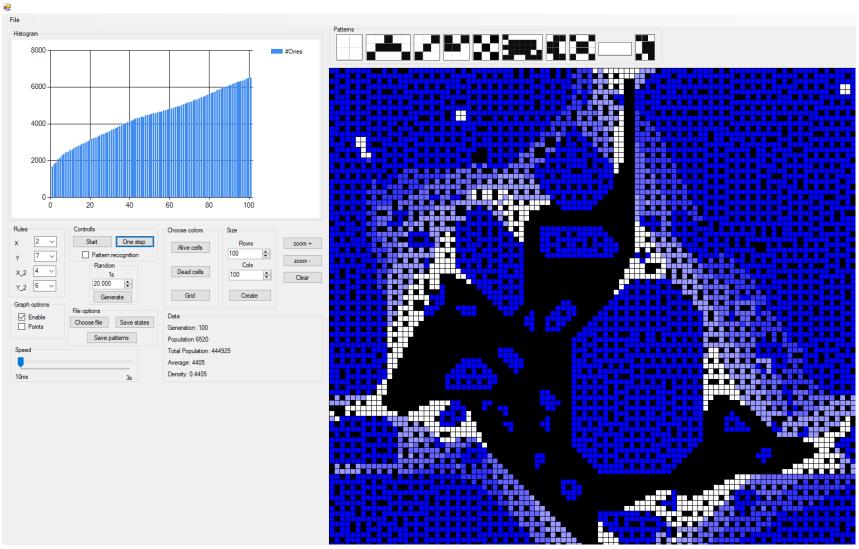
Figure 1.23: Grafos según la regla "Diffusion" archivo filtrado matriz 4x4



Podemos notar entre ambas reglas que hay varios grafos similares, pero quizá son un poco más complejas las figuras que son encontradas aplicando la regla de "Diffusion"

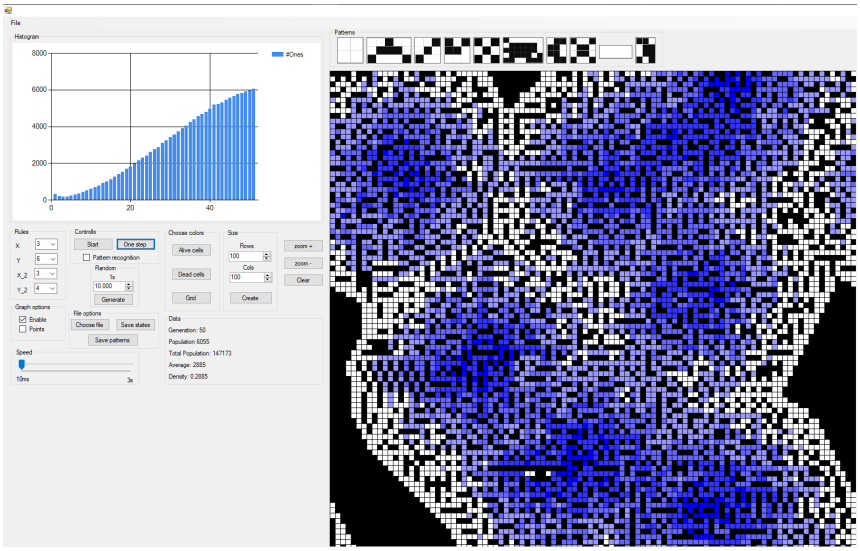
También se han probado más reglas en este autómata, primeramente se probó con la regla 2, 7, 4, 6 y una distribución de 20% de probabilidad en un espacio de cien por cien células. Además podemos apreciar el resultado al llegar a las 100 generaciones

Figure 1.24: Ejecución según la regla 2, 7, 4, 6



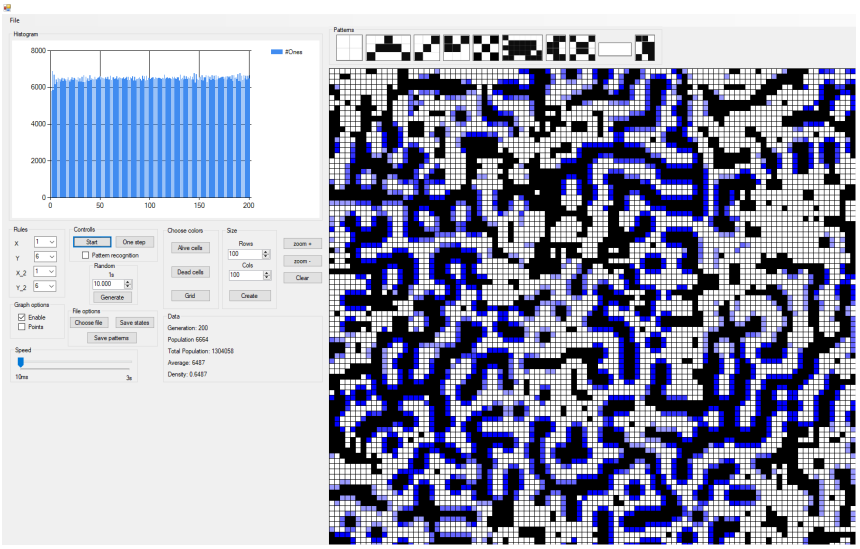
Otra regla que fue probada es la 3, 6, 3, 4 podemos ver que el comportamiento es similar a la anterior, y podemos apreciar esto en la gráfica, solo que para este caso se ejecutó el programa hasta la generación cincuenta y con una densidad en la distribución de células vivas menor.

Figure 1.25: Ejecución según la regla 3, 6, 3, 4



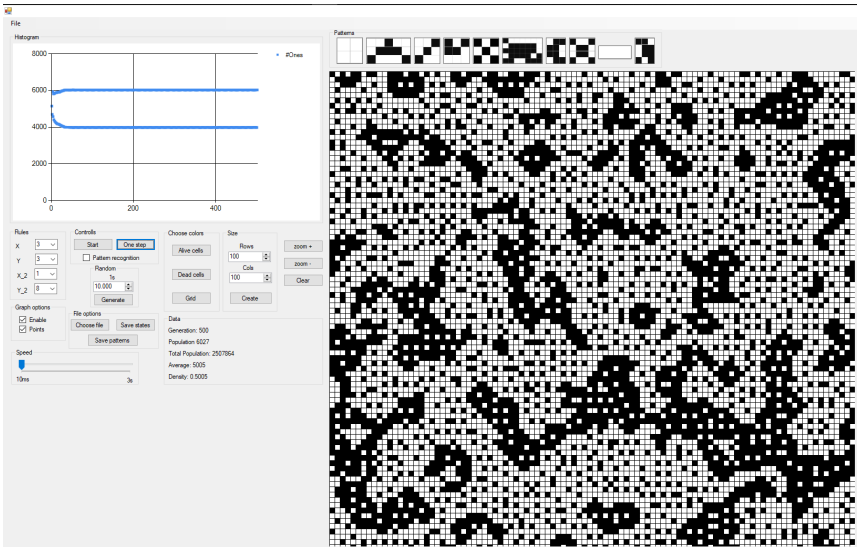
La siguiente regla que fue probada es la 1, 6, 1, 6 esta tiene un crecimiento bastante curioso, fue probada hasta la generación 200, y gracias a la función que se añadió del cambio de color cuando las células se han "estancado" podemos ver claramente que el comportamiento con esta regla no es de un oscilador.

Figure 1.26: Ejecución según la regla 1, 6, 1, 6



La siguiente regla probada es 3, 3, 1, 8 esta fue ejecutada hasta la generación 500, y podemos observar claramente que el comportamiento es de un oscilador, esto es fácil de notar gracias a la gráfica que tenemos en la parte superior izquierda, además de que no hay células "estancadas"

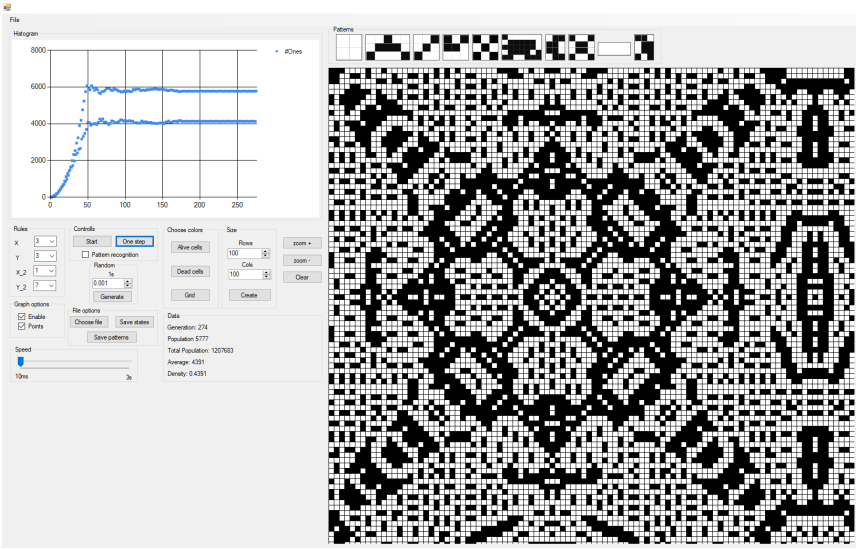
Figure 1.27: Ejecución según la regla 3, 3, 1, 8



La siguiente regla probada es 3, 3, 1, 7 también es muy interesante esta regla, ya que con solo poner un punto se llena el espacio completamente, además de no tener células estancadas y además comportarse como un oscilador cuando el espacio

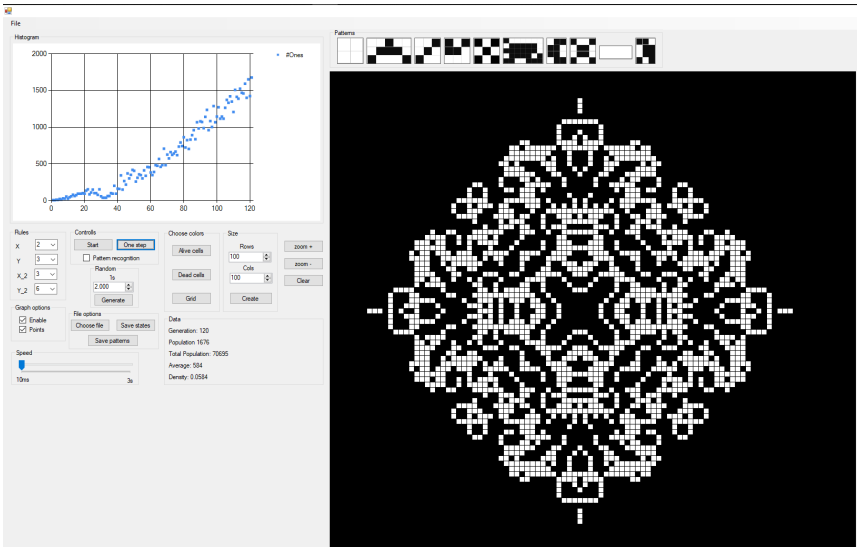
está lleno. La configuración inicial para esta regla fue un punto cerca del centro del espacio.

Figure 1.28: Ejecución según la regla 3, 3, 1, 7



Finalmente última regla que fue ingresada fue la 2, 3, 3, 6 la cual fue probada con una configuración inicial en específico, para la imagen que se muestra a continuación se inició con una línea horizontal de cinco células, pero también el comportamiento es diferente al colocar una de tres células, se comporta igual que en la regla de "Game of life", si añadimos una célula más se "estancan" las células

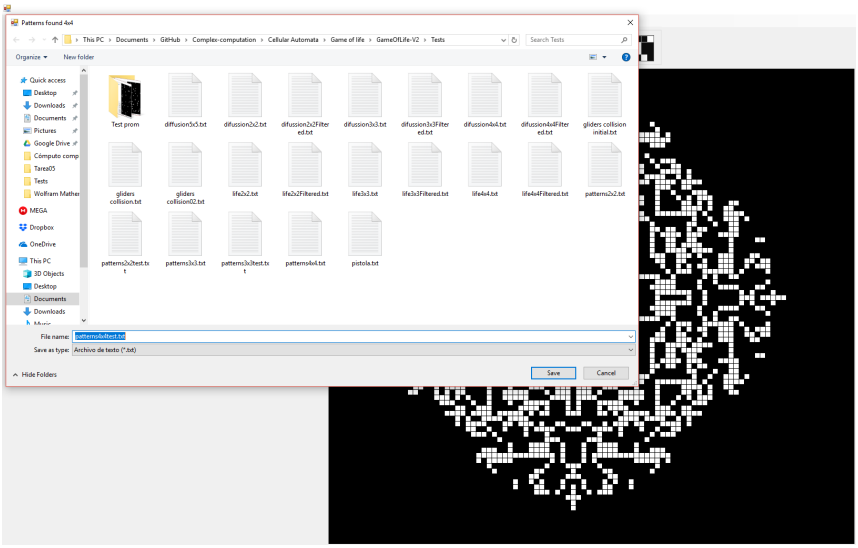
Figure 1.29: Ejecución según la regla 2, 3, 3, 6



1.7 Reconocimiento de patrones

Como se ha mencionado previamente, se ha creado un pequeño algoritmo que hace de reconocimiento de patrones, para hacer el reconocimiento de las submatrices en el programa solo es necesario activar el pequeño check box con la leyenda "Pattern recognition", una ves que se ha activado el algoritmo evalúa pequeñas submatrices desde dos por dos hasta cuatro por cuatro, esto debido a que el sistema operativo no fue de gran ayuda al asignar más memoria RAM a la aplicación, limitándonos a solo dos gigabytes. Si el check box está activado, se ha ejecutado el programa y se selecciona el botón con la leyenda "Save patterns" se guardará el registro de los patrones encontrados desde que el check box fue seleccionado hasta el momento en el que se dió click al botón antes mencionado.

Figure 1.30: Almacenamiento de los patrones

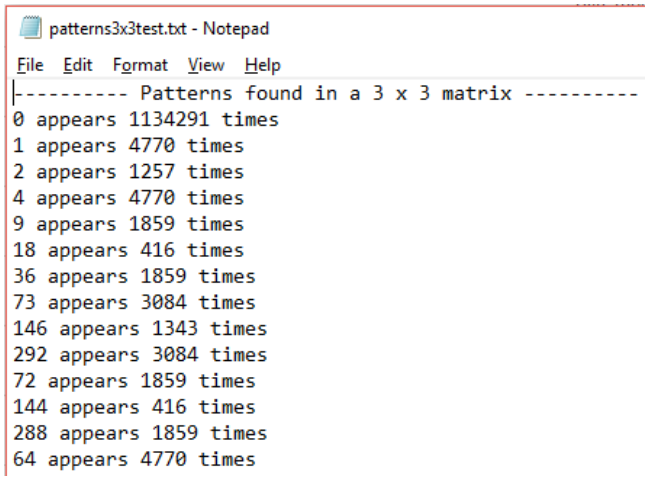


Una ves que se ha presionado el botón con la leyenda "Save patterns" se guardarán tres archivos, uno por cada tamaño de las submatrices evaluadas, en la siguiente imagen podemos apreciar los tres archivos generados

Figure 1.31: Archivos generados

Y en la siguiente imagen podemos ver el contenido generado al evaluar las submatrices con dimensión tres por tres

Figure 1.32: fragmento del contenido del archivo patterns3x3test.txt



1.7.1 Código

A continuación se aneza el código generado para la creación de este simulador. Primeramente se muestra el código de una clase que se ha creado para la manipulación de colores dentro de la aplicación, ya que a diferencia de la versión anterior ahora es posible elegir cualquier color existente en la paleta de colores RGB

```
1  using System;
2  using System.Collections.Generic;
3  using System.Drawing;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GameOfLife
9  {
10     /// <summary>
11     /// This class has been made to manipulate colors
12     /// </summary>
13     public static class ColorHandler
14     {
15         /// <summary>
16         /// Convert from a color objet to an unsigned int
17         /// </summary>
18         /// <param name="color">Source color</param>
19         /// <returns>unsigned int value of the color</returns>
20         public static uint fromColorToInt(Color color)
21         {
22             return (uint)((color.A << 24) | (color.R << 16) | (color.G
                   << 8) | (color.B << 0));
```

```
23          }
24          /// <summary>
25          /// Convert from an unsigned int to a color object
26          /// </summary>
27          /// <param name="argb">Unsigned int value of a color</param>
28          /// <returns>A color object</returns>
29          public static Color fromIntToColor(uint argb)
30          {
31              byte[] bytes = BitConverter.GetBytes(argb);
32              return Color.FromArgb(bytes[2], bytes[1], bytes[0]);
33          }
34
35          public static Color fromIntToGradient(uint code, uint base_color
                ) {
36              try
37              {
38                  if (code <= base_color && code > base_color - 10)
39                  {
40                      return fromIntToColor(base_color);
41                  }
42                  else if (code <= base_color - 10 && code > base_color -
                        20)
43                  {
44                      return Color.FromArgb(255, 153, 153, 255);
45                  }
46                  else if (code <= base_color - 20 && code > base_color -
                        30)
47                  {
48                      return Color.FromArgb(255, 102, 102, 255);
49                  }
50                  else if (code <= base_color - 30 && code > base_color -
                        40)
51                  {
52                      return Color.FromArgb(255, 51, 51, 255);
53                  }
54                  else
55                  {
56                      return Color.FromArgb(255, 0, 0, 255);
57                  }
58              }
59              catch (Exception) {
60                  return fromIntToColor(base_color);
61              }
62          }
63      }
64  }
```

A continuación se muestra una clase llamada Graph, la cual fue creada con la finalidad de permitir "moldear" los grafos que eran encontrados a partir de todas las combinaciones generadas en cada submatriz

```
1  using System;
2  using System.Collections.Generic;
```

```
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GameOfLife
8  {
9      class Graph
10     {
11         private Dictionary<ulong, List<ulong>> nodes;
12
13         public Graph(Dictionary<ulong, List<ulong>> init_nodes) {
14             nodes = new Dictionary<ulong, List<ulong>>(init_nodes);
15         }
16
17         public void addNodes(ulong key, List<ulong> value) {
18             nodes.Add(key, value);
19         }
20
21         public ulong getTotalVerticesPerNode() {
22             ulong vertices = 0;
23             foreach (KeyValuePair<ulong, List<ulong>> item in nodes) {
24                 vertices += (ulong)item.Value.Count;
25             }
26             return vertices;
27         }
28
29         public ulong getKeyInt() {
30             return getTotalVerticesPerNode();
31         }
32         public string getKey() {
33
34             Dictionary<ulong, ulong> pre_key = new Dictionary<ulong,
                   ulong>();
35             foreach (KeyValuePair<ulong, List<ulong>> item in nodes) {
36
37                 if (!pre_key.ContainsKey(item.Key))
38                     pre_key.Add(item.Key, 1);
39
40                 for (int i = 0; i < item.Value.Count; i++) {
41                     if (pre_key.ContainsKey(item.Value[i]))
42                     {
43                         pre_key[item.Value[i]]++;
44                     }
45                     else {
46                         pre_key.Add(item.Value[i], 1);
47                     }
48                 }
49             }
50
51             string key = "";
52             foreach (KeyValuePair<ulong, ulong> item in pre_key) {
53                 key += item.Value + ((pre_key.Last().Key == item.Key) ?
                       "" : ",");
```

```
54              }
55              return key;
56          }
57
58          private void printList(List<ulong> list) {
59              Console.WriteLine();
60              for (int i = 0; i < list.Count; i++) {
61                  Console.Write(list[i] + " ");
62              }
63              Console.WriteLine();
64          }
65          public Dictionary<ulong, List<ulong>> getAllNodes() {
66              return nodes;
67          }
68
69          public void printStates() {
70              Console.WriteLine("Printing states");
71              foreach (KeyValuePair<ulong, List<ulong>> item in nodes) {
72                  Console.Write(item.Key + " -> ");
73                  for (int i = 0; i < item.Value.Count; i++) {
74                      Console.Write(item.Value[i] + " ");
75                  }
76                  Console.WriteLine();
77              }
78          }
79
80          public void toMathematica() {
81              foreach (KeyValuePair<ulong, List<ulong>> item in nodes)
82              {
83                  for (int i = 0; i < item.Value.Count; i++)
84                  {
85                      Console.Write(item.Value[i] + " -> " + item.Key + ",
                            ");
86                  }
87              }
88          }
89      }
90  }
```

A continuación se anexa el código generado para la creación del simulador. Primero tenemos la siguiente clase la cual contiene código autogenerado por el IDE. Esta código es únicamente para el manejo de la interfaz.

```
1  namespace GameOfLife
2  {
3      partial class Form1
4      {
5          /// <summary>
6          /// Variable del ñdiseador necesaria.
7          /// </summary>
8          private System.ComponentModel.IContainer components = null;
9
10         /// <summary>
```

```
11          /// Limpiar los recursos que se éestn usando.
12          /// </summary>
13          /// <param name="disposing">true si los recursos administrados
                se deben desechar; false en caso contrario.</param>
14          protected override void Dispose(bool disposing)
15          {
16              if (disposing && (components != null))
17              {
18                  components.Dispose();
19              }
20              base.Dispose(disposing);
21          }
22
23          #region óCdigo generado por el ñDiseador de Windows Forms
24
25          /// <summary>
26          /// éMtodo necesario para admitir el ñDiseador. No se puede
                modificar
27          /// el contenido de este émtodo con el editor de ócdigo.
28          /// </summary>
29          private void InitializeComponent()
30          {
31              this.components = new System.ComponentModel.Container();
32              System.Windows.Forms.DataVisualization.Charting.ChartArea
                    chartArea2 = new System.Windows.Forms.DataVisualization.
                    Charting.ChartArea();
33              System.Windows.Forms.DataVisualization.Charting.Legend
                    legend2 = new System.Windows.Forms.DataVisualization.
                    Charting.Legend();
34              System.Windows.Forms.DataVisualization.Charting.Series
                    series2 = new System.Windows.Forms.DataVisualization.
                    Charting.Series();
35              System.ComponentModel.ComponentResourceManager resources =
                    new System.ComponentModel.ComponentResourceManager(
                    typeof(Form1));
36              this.PBAutomataSimulator = new System.Windows.Forms.
                    PictureBox();
37              this.CHHistogram = new System.Windows.Forms.
                    DataVisualization.Charting.Chart();
38              this.groupBox1 = new System.Windows.Forms.GroupBox();
39              this.BTNStart = new System.Windows.Forms.Button();
40              this.groupBox2 = new System.Windows.Forms.GroupBox();
41              this.label5 = new System.Windows.Forms.Label();
42              this.label4 = new System.Windows.Forms.Label();
43              this.label3 = new System.Windows.Forms.Label();
44              this.label2 = new System.Windows.Forms.Label();
45              this.ComboBY2i = new System.Windows.Forms.ComboBox();
46              this.ComboBX2i = new System.Windows.Forms.ComboBox();
47              this.ComboBYi = new System.Windows.Forms.ComboBox();
48              this.ComboBXi = new System.Windows.Forms.ComboBox();
49              this.TXTGeneration = new System.Windows.Forms.Label();
50              this.TXTPopulation = new System.Windows.Forms.Label();
51              this.BTNStep = new System.Windows.Forms.Button();
```

```
52              this.TBSpeed = new System.Windows.Forms.TrackBar();
53              this.TimerSimulation = new System.Windows.Forms.Timer(this.
                    components);
54              this.flowLayoutPanel1 = new System.Windows.Forms.
                    FlowLayoutPanel();
55              this.groupBox3 = new System.Windows.Forms.GroupBox();
56              this.label6 = new System.Windows.Forms.Label();
57              this.label1 = new System.Windows.Forms.Label();
58              this.BTNZoomP = new System.Windows.Forms.Button();
59              this.BTNZoomM = new System.Windows.Forms.Button();
60              this.groupBox4 = new System.Windows.Forms.GroupBox();
61              this.groupBox5 = new System.Windows.Forms.GroupBox();
62              this.button1 = new System.Windows.Forms.Button();
63              this.numericOnes = new System.Windows.Forms.NumericUpDown();
64              this.label7 = new System.Windows.Forms.Label();
65              this.groupBox6 = new System.Windows.Forms.GroupBox();
66              this.BTNGrid = new System.Windows.Forms.Button();
67              this.BTNDeadCells = new System.Windows.Forms.Button();
68              this.BTNAliveCells = new System.Windows.Forms.Button();
69              this.BTNSelectFile = new System.Windows.Forms.Button();
70              this.BTNClear = new System.Windows.Forms.Button();
71              this.groupBox7 = new System.Windows.Forms.GroupBox();
72              this.BTNCreateMatrix = new System.Windows.Forms.Button();
73              this.numericCols = new System.Windows.Forms.NumericUpDown();
74              this.numericRows = new System.Windows.Forms.NumericUpDown();
75              this.label11 = new System.Windows.Forms.Label();
76              this.label8 = new System.Windows.Forms.Label();
77              this.groupBox8 = new System.Windows.Forms.GroupBox();
78              this.BTNSave = new System.Windows.Forms.Button();
79              this.label12 = new System.Windows.Forms.Label();
80              this.label13 = new System.Windows.Forms.Label();
81              this.label14 = new System.Windows.Forms.Label();
82              this.groupBox9 = new System.Windows.Forms.GroupBox();
83              this.CBPoints = new System.Windows.Forms.CheckBox();
84              this.CheckGraphEnabled = new System.Windows.Forms.CheckBox()
                    ;
85              this.groupBox10 = new System.Windows.Forms.GroupBox();
86              this.pictureBox10 = new System.Windows.Forms.PictureBox();
87              this.pictureBox9 = new System.Windows.Forms.PictureBox();
88              this.pictureBox8 = new System.Windows.Forms.PictureBox();
89              this.pictureBox7 = new System.Windows.Forms.PictureBox();
90              this.pictureBox6 = new System.Windows.Forms.PictureBox();
91              this.pictureBox5 = new System.Windows.Forms.PictureBox();
92              this.pictureBox4 = new System.Windows.Forms.PictureBox();
93              this.pictureBox3 = new System.Windows.Forms.PictureBox();
94              this.pictureBox2 = new System.Windows.Forms.PictureBox();
95              this.pictureBox1 = new System.Windows.Forms.PictureBox();
96              this.groupBox11 = new System.Windows.Forms.GroupBox();
97              this.colorDialog = new System.Windows.Forms.ColorDialog();
98              this.menuStrip1 = new System.Windows.Forms.MenuStrip();
99              this.generarPatronesToolStripMenuItem = new System.Windows.
                    Forms.ToolStripMenuItem();
100             this.generatePatternsToolStripMenuItem = new System.Windows.
```

```
            Forms.ToolStripMenuItem();
101         this.BTNSavePatterns = new System.Windows.Forms.Button();
102         this.CBPatternRecognition = new System.Windows.Forms.
                CheckBox();
103         ((System.ComponentModel.ISupportInitialize)(this.
                PBAutomataSimulator)).BeginInit();
104         ((System.ComponentModel.ISupportInitialize)(this.CHHistogram
                )).BeginInit();
105         this.groupBox1.SuspendLayout();
106         this.groupBox2.SuspendLayout();
107         ((System.ComponentModel.ISupportInitialize)(this.TBSpeed)).
                BeginInit();
108         this.flowLayoutPanel1.SuspendLayout();
109         this.groupBox3.SuspendLayout();
110         this.groupBox4.SuspendLayout();
111         this.groupBox5.SuspendLayout();
112         ((System.ComponentModel.ISupportInitialize)(this.numericOnes
                )).BeginInit();
113         this.groupBox6.SuspendLayout();
114         this.groupBox7.SuspendLayout();
115         ((System.ComponentModel.ISupportInitialize)(this.numericCols
                )).BeginInit();
116         ((System.ComponentModel.ISupportInitialize)(this.numericRows
                )).BeginInit();
117         this.groupBox8.SuspendLayout();
118         this.groupBox9.SuspendLayout();
119         this.groupBox10.SuspendLayout();
120         ((System.ComponentModel.ISupportInitialize)(this.
                pictureBox10)).BeginInit();
121         ((System.ComponentModel.ISupportInitialize)(this.pictureBox9
                )).BeginInit();
122         ((System.ComponentModel.ISupportInitialize)(this.pictureBox8
                )).BeginInit();
123         ((System.ComponentModel.ISupportInitialize)(this.pictureBox7
                )).BeginInit();
124         ((System.ComponentModel.ISupportInitialize)(this.pictureBox6
                )).BeginInit();
125         ((System.ComponentModel.ISupportInitialize)(this.pictureBox5
                )).BeginInit();
126         ((System.ComponentModel.ISupportInitialize)(this.pictureBox4
                )).BeginInit();
127         ((System.ComponentModel.ISupportInitialize)(this.pictureBox3
                )).BeginInit();
128         ((System.ComponentModel.ISupportInitialize)(this.pictureBox2
                )).BeginInit();
129         ((System.ComponentModel.ISupportInitialize)(this.pictureBox1
                )).BeginInit();
130         this.groupBox11.SuspendLayout();
131         this.menuStrip1.SuspendLayout();
132         this.SuspendLayout();
133         //
134         // PBAutomataSimulator
135         //
```

```
136             this.PBAutomataSimulator.BackColor = System.Drawing.
                    SystemColors.ActiveCaptionText;
137             this.PBAutomataSimulator.Location = new System.Drawing.Point
                    (3, 3);
138             this.PBAutomataSimulator.Name = "PBAutomataSimulator";
139             this.PBAutomataSimulator.Size = new System.Drawing.Size(545,
                     505);
140             this.PBAutomataSimulator.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.AutoSize;
141             this.PBAutomataSimulator.TabIndex = 1;
142             this.PBAutomataSimulator.TabStop = false;
143             this.PBAutomataSimulator.Paint += new System.Windows.Forms.
                    PaintEventHandler(this.PBAutomataSimulator_Paint);
144             this.PBAutomataSimulator.MouseDown += new System.Windows.
                    Forms.MouseEventHandler(this.
                    PBAutomataSimulator_MouseDown);
145             this.PBAutomataSimulator.MouseMove += new System.Windows.
                    Forms.MouseEventHandler(this.
                    PBAutomataSimulator_MouseMove);
146             //
147             // CHHistogram
148             //
149             chartArea2.Name = "ChartArea1";
150             this.CHHistogram.ChartAreas.Add(chartArea2);
151             legend2.Name = "Legend1";
152             this.CHHistogram.Legends.Add(legend2);
153             this.CHHistogram.Location = new System.Drawing.Point(6, 19);
154             this.CHHistogram.Name = "CHHistogram";
155             series2.ChartArea = "ChartArea1";
156             series2.Legend = "Legend1";
157             series2.Name = "#Ones";
158             this.CHHistogram.Series.Add(series2);
159             this.CHHistogram.Size = new System.Drawing.Size(584, 337);
160             this.CHHistogram.TabIndex = 2;
161             this.CHHistogram.Text = "chart1";
162             //
163             // groupBox1
164             //
165             this.groupBox1.Controls.Add(this.CHHistogram);
166             this.groupBox1.Location = new System.Drawing.Point(13, 31);
167             this.groupBox1.Name = "groupBox1";
168             this.groupBox1.Size = new System.Drawing.Size(596, 362);
169             this.groupBox1.TabIndex = 3;
170             this.groupBox1.TabStop = false;
171             this.groupBox1.Text = "Histogram";
172             //
173             // BTNStart
174             //
175             this.BTNStart.Location = new System.Drawing.Point(6, 19);
176             this.BTNStart.Name = "BTNStart";
177             this.BTNStart.Size = new System.Drawing.Size(75, 23);
178             this.BTNStart.TabIndex = 4;
179             this.BTNStart.Text = "Start";
```

```
180             this.BTNStart.UseVisualStyleBackColor = true;
181             this.BTNStart.Click += new System.EventHandler(this.
                    BTNStart_Click);
182             //
183             // groupBox2
184             //
185             this.groupBox2.Controls.Add(this.label5);
186             this.groupBox2.Controls.Add(this.label4);
187             this.groupBox2.Controls.Add(this.label3);
188             this.groupBox2.Controls.Add(this.label2);
189             this.groupBox2.Controls.Add(this.ComboBY2i);
190             this.groupBox2.Controls.Add(this.ComboBX2i);
191             this.groupBox2.Controls.Add(this.ComboBYi);
192             this.groupBox2.Controls.Add(this.ComboBXi);
193             this.groupBox2.Location = new System.Drawing.Point(13, 402);
194             this.groupBox2.Name = "groupBox2";
195             this.groupBox2.Size = new System.Drawing.Size(107, 137);
196             this.groupBox2.TabIndex = 6;
197             this.groupBox2.TabStop = false;
198             this.groupBox2.Text = "Rules";
199             //
200             // label5
201             //
202             this.label5.AutoSize = true;
203             this.label5.Location = new System.Drawing.Point(11, 111);
204             this.label5.Name = "label5";
205             this.label5.Size = new System.Drawing.Size(26, 13);
206             this.label5.TabIndex = 7;
207             this.label5.Text = "Y_2";
208             //
209             // label4
210             //
211             this.label4.AutoSize = true;
212             this.label4.Location = new System.Drawing.Point(11, 83);
213             this.label4.Name = "label4";
214             this.label4.Size = new System.Drawing.Size(26, 13);
215             this.label4.TabIndex = 6;
216             this.label4.Text = "X_2";
217             //
218             // label3
219             //
220             this.label3.AutoSize = true;
221             this.label3.Location = new System.Drawing.Point(10, 55);
222             this.label3.Name = "label3";
223             this.label3.Size = new System.Drawing.Size(14, 13);
224             this.label3.TabIndex = 5;
225             this.label3.Text = "Y";
226             //
227             // label2
228             //
229             this.label2.AutoSize = true;
230             this.label2.Location = new System.Drawing.Point(8, 27);
231             this.label2.Name = "label2";
```

```
232             this.label2.Size = new System.Drawing.Size(14, 13);
233             this.label2.TabIndex = 4;
234             this.label2.Text = "X";
235             //
236             // ComboBY2i
237             //
238             this.ComboBY2i.FormattingEnabled = true;
239             this.ComboBY2i.Items.AddRange(new object[] {
240             "1",
241             "2",
242             "3",
243             "4",
244             "5",
245             "6",
246             "7",
247             "8"});
248             this.ComboBY2i.Location = new System.Drawing.Point(47, 104);
249             this.ComboBY2i.Name = "ComboBY2i";
250             this.ComboBY2i.Size = new System.Drawing.Size(43, 21);
251             this.ComboBY2i.TabIndex = 3;
252             //
253             // ComboBX2i
254             //
255             this.ComboBX2i.FormattingEnabled = true;
256             this.ComboBX2i.Items.AddRange(new object[] {
257             "1",
258             "2",
259             "3",
260             "4",
261             "5",
262             "6",
263             "7",
264             "8"});
265             this.ComboBX2i.Location = new System.Drawing.Point(47, 76);
266             this.ComboBX2i.Name = "ComboBX2i";
267             this.ComboBX2i.Size = new System.Drawing.Size(43, 21);
268             this.ComboBX2i.TabIndex = 2;
269             //
270             // ComboBYi
271             //
272             this.ComboBYi.FormattingEnabled = true;
273             this.ComboBYi.Items.AddRange(new object[] {
274             "1",
275             "2",
276             "3",
277             "4",
278             "5",
279             "6",
280             "7",
281             "8"});
282             this.ComboBYi.Location = new System.Drawing.Point(48, 48);
283             this.ComboBYi.Name = "ComboBYi";
284             this.ComboBYi.Size = new System.Drawing.Size(43, 21);
```

```
285             this.ComboBYi.TabIndex = 1;
286             //
287             // ComboBXi
288             //
289             this.ComboBXi.FormattingEnabled = true;
290             this.ComboBXi.Items.AddRange(new object[] {
291             "1",
292             "2",
293             "3",
294             "4",
295             "5",
296             "6",
297             "7",
298             "8"});
299             this.ComboBXi.Location = new System.Drawing.Point(48, 20);
300             this.ComboBXi.Name = "ComboBXi";
301             this.ComboBXi.Size = new System.Drawing.Size(43, 21);
302             this.ComboBXi.TabIndex = 0;
303             //
304             // TXTGeneration
305             //
306             this.TXTGeneration.AutoSize = true;
307             this.TXTGeneration.BackColor = System.Drawing.Color.
                    Transparent;
308             this.TXTGeneration.ForeColor = System.Drawing.Color.Black;
309             this.TXTGeneration.Location = new System.Drawing.Point(6,
                    22);
310             this.TXTGeneration.Name = "TXTGeneration";
311             this.TXTGeneration.Size = new System.Drawing.Size(62, 13);
312             this.TXTGeneration.TabIndex = 7;
313             this.TXTGeneration.Text = "Generation ";
314             //
315             // TXTPopulation
316             //
317             this.TXTPopulation.AutoSize = true;
318             this.TXTPopulation.Location = new System.Drawing.Point(6,
                    44);
319             this.TXTPopulation.Name = "TXTPopulation";
320             this.TXTPopulation.Size = new System.Drawing.Size(57, 13);
321             this.TXTPopulation.TabIndex = 8;
322             this.TXTPopulation.Text = "Population";
323             //
324             // BTNStep
325             //
326             this.BTNStep.Location = new System.Drawing.Point(87, 19);
327             this.BTNStep.Name = "BTNStep";
328             this.BTNStep.Size = new System.Drawing.Size(75, 23);
329             this.BTNStep.TabIndex = 9;
330             this.BTNStep.Text = "One step";
331             this.BTNStep.UseVisualStyleBackColor = true;
332             this.BTNStep.Click += new System.EventHandler(this.
                    BTNStep_Click);
333             //
```

```
334             // TBSpeed
335             //
336             this.TBSpeed.Location = new System.Drawing.Point(6, 19);
337             this.TBSpeed.Maximum = 3000;
338             this.TBSpeed.Minimum = 10;
339             this.TBSpeed.Name = "TBSpeed";
340             this.TBSpeed.Size = new System.Drawing.Size(233, 45);
341             this.TBSpeed.TabIndex = 10;
342             this.TBSpeed.Value = 1000;
343             this.TBSpeed.ValueChanged += new System.EventHandler(this.
                    trackBar1_ValueChanged);
344             //
345             // TimerSimulation
346             //
347             this.TimerSimulation.Tick += new System.EventHandler(this.
                    TimerSimulation_Tick);
348             //
349             // flowLayoutPanel1
350             //
351             this.flowLayoutPanel1.Anchor = ((System.Windows.Forms.
                    AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
                     System.Windows.Forms.AnchorStyles.Bottom)
352             | System.Windows.Forms.AnchorStyles.Left)
353             | System.Windows.Forms.AnchorStyles.Right)));
354             this.flowLayoutPanel1.AutoScroll = true;
355             this.flowLayoutPanel1.AutoSizeMode = System.Windows.Forms.
                    AutoSizeMode.GrowAndShrink;
356             this.flowLayoutPanel1.Controls.Add(this.PBAutomataSimulator)
                    ;
357             this.flowLayoutPanel1.Location = new System.Drawing.Point
                    (618, 100);
358             this.flowLayoutPanel1.MinimumSize = new System.Drawing.Size
                    (639, 600);
359             this.flowLayoutPanel1.Name = "flowLayoutPanel1";
360             this.flowLayoutPanel1.Size = new System.Drawing.Size(639,
                    600);
361             this.flowLayoutPanel1.TabIndex = 11;
362             //
363             // groupBox3
364             //
365             this.groupBox3.Controls.Add(this.label6);
366             this.groupBox3.Controls.Add(this.label1);
367             this.groupBox3.Controls.Add(this.TBSpeed);
368             this.groupBox3.Location = new System.Drawing.Point(16, 631);
369             this.groupBox3.Name = "groupBox3";
370             this.groupBox3.Size = new System.Drawing.Size(242, 69);
371             this.groupBox3.TabIndex = 12;
372             this.groupBox3.TabStop = false;
373             this.groupBox3.Text = "Speed";
374             //
375             // label6
376             //
377             this.label6.AutoSize = true;
```

```
378             this.label6.Location = new System.Drawing.Point(217, 53);
379             this.label6.Name = "label6";
380             this.label6.Size = new System.Drawing.Size(18, 13);
381             this.label6.TabIndex = 12;
382             this.label6.Text = "3s";
383             //
384             // label1
385             //
386             this.label1.AutoSize = true;
387             this.label1.Location = new System.Drawing.Point(6, 51);
388             this.label1.Name = "label1";
389             this.label1.Size = new System.Drawing.Size(32, 13);
390             this.label1.TabIndex = 11;
391             this.label1.Text = "10ms";
392             //
393             // BTNZoomP
394             //
395             this.BTNZoomP.Location = new System.Drawing.Point(534, 424);
396             this.BTNZoomP.Name = "BTNZoomP";
397             this.BTNZoomP.Size = new System.Drawing.Size(75, 23);
398             this.BTNZoomP.TabIndex = 13;
399             this.BTNZoomP.Text = "zoom +";
400             this.BTNZoomP.UseVisualStyleBackColor = true;
401             this.BTNZoomP.Click += new System.EventHandler(this.
                    BTNZoomP_Click);
402             //
403             // BTNZoomM
404             //
405             this.BTNZoomM.Location = new System.Drawing.Point(534, 457);
406             this.BTNZoomM.Name = "BTNZoomM";
407             this.BTNZoomM.Size = new System.Drawing.Size(75, 23);
408             this.BTNZoomM.TabIndex = 14;
409             this.BTNZoomM.Text = "zoom -";
410             this.BTNZoomM.UseVisualStyleBackColor = true;
411             this.BTNZoomM.Click += new System.EventHandler(this.
                    BTNZoomM_Click);
412             //
413             // groupBox4
414             //
415             this.groupBox4.Controls.Add(this.CBPatternRecognition);
416             this.groupBox4.Controls.Add(this.BTNStart);
417             this.groupBox4.Controls.Add(this.BTNStep);
418             this.groupBox4.Location = new System.Drawing.Point(128, 402)
                    ;
419             this.groupBox4.Name = "groupBox4";
420             this.groupBox4.Size = new System.Drawing.Size(168, 69);
421             this.groupBox4.TabIndex = 15;
422             this.groupBox4.TabStop = false;
423             this.groupBox4.Text = "Controlls";
424             //
425             // groupBox5
426             //
427             this.groupBox5.Controls.Add(this.button1);
```

```
428             this.groupBox5.Controls.Add(this.numericOnes);
429             this.groupBox5.Controls.Add(this.label7);
430             this.groupBox5.Location = new System.Drawing.Point(166, 471)
                    ;
431             this.groupBox5.Name = "groupBox5";
432             this.groupBox5.Size = new System.Drawing.Size(92, 82);
433             this.groupBox5.TabIndex = 16;
434             this.groupBox5.TabStop = false;
435             this.groupBox5.Text = "Random";
436             //
437             // button1
438             //
439             this.button1.Location = new System.Drawing.Point(7, 55);
440             this.button1.Name = "button1";
441             this.button1.Size = new System.Drawing.Size(75, 23);
442             this.button1.TabIndex = 4;
443             this.button1.Text = "Generate";
444             this.button1.UseVisualStyleBackColor = true;
445             this.button1.Click += new System.EventHandler(this.
                    button1_Click);
446             //
447             // numericOnes
448             //
449             this.numericOnes.DecimalPlaces = 3;
450             this.numericOnes.Location = new System.Drawing.Point(6, 32);
451             this.numericOnes.Name = "numericOnes";
452             this.numericOnes.Size = new System.Drawing.Size(76, 20);
453             this.numericOnes.TabIndex = 2;
454             //
455             // label7
456             //
457             this.label7.AutoSize = true;
458             this.label7.Location = new System.Drawing.Point(33, 17);
459             this.label7.Name = "label7";
460             this.label7.Size = new System.Drawing.Size(18, 13);
461             this.label7.TabIndex = 0;
462             this.label7.Text = "1s";
463             //
464             // groupBox6
465             //
466             this.groupBox6.Controls.Add(this.BTNGrid);
467             this.groupBox6.Controls.Add(this.BTNDeadCells);
468             this.groupBox6.Controls.Add(this.BTNAliveCells);
469             this.groupBox6.Location = new System.Drawing.Point(306, 404)
                    ;
470             this.groupBox6.Name = "groupBox6";
471             this.groupBox6.Size = new System.Drawing.Size(106, 157);
472             this.groupBox6.TabIndex = 17;
473             this.groupBox6.TabStop = false;
474             this.groupBox6.Text = "Choose colors";
475             //
476             // BTNGrid
477             //
```

```
478             this.BTNGrid.Location = new System.Drawing.Point(15, 119);
479             this.BTNGrid.Name = "BTNGrid";
480             this.BTNGrid.Size = new System.Drawing.Size(75, 23);
481             this.BTNGrid.TabIndex = 7;
482             this.BTNGrid.Text = "Grid";
483             this.BTNGrid.UseVisualStyleBackColor = true;
484             this.BTNGrid.Click += new System.EventHandler(this.
                    BTNGrid_Click);
485             //
486             // BTNDeadCells
487             //
488             this.BTNDeadCells.Location = new System.Drawing.Point(15,
                    74);
489             this.BTNDeadCells.Name = "BTNDeadCells";
490             this.BTNDeadCells.Size = new System.Drawing.Size(75, 23);
491             this.BTNDeadCells.TabIndex = 6;
492             this.BTNDeadCells.Text = "Dead cells";
493             this.BTNDeadCells.UseVisualStyleBackColor = true;
494             this.BTNDeadCells.Click += new System.EventHandler(this.
                    BTNDeadCells_Click);
495             //
496             // BTNAliveCells
497             //
498             this.BTNAliveCells.Location = new System.Drawing.Point(15,
                    28);
499             this.BTNAliveCells.Name = "BTNAliveCells";
500             this.BTNAliveCells.Size = new System.Drawing.Size(75, 23);
501             this.BTNAliveCells.TabIndex = 5;
502             this.BTNAliveCells.Text = "Alive cells";
503             this.BTNAliveCells.UseVisualStyleBackColor = true;
504             this.BTNAliveCells.Click += new System.EventHandler(this.
                    button2_Click_1);
505             //
506             // BTNSelectFile
507             //
508             this.BTNSelectFile.Location = new System.Drawing.Point(3,
                    17);
509             this.BTNSelectFile.Name = "BTNSelectFile";
510             this.BTNSelectFile.Size = new System.Drawing.Size(75, 23);
511             this.BTNSelectFile.TabIndex = 18;
512             this.BTNSelectFile.Text = "Choose file";
513             this.BTNSelectFile.UseVisualStyleBackColor = true;
514             this.BTNSelectFile.Click += new System.EventHandler(this.
                    button2_Click);
515             //
516             // BTNClear
517             //
518             this.BTNClear.Location = new System.Drawing.Point(534, 489);
519             this.BTNClear.Name = "BTNClear";
520             this.BTNClear.Size = new System.Drawing.Size(75, 23);
521             this.BTNClear.TabIndex = 19;
522             this.BTNClear.Text = "Clear";
523             this.BTNClear.UseVisualStyleBackColor = true;
```

```
524             this.BTNClear.Click += new System.EventHandler(this.
                    BTNClear_Click);
525             //
526             // groupBox7
527             //
528             this.groupBox7.Controls.Add(this.BTNCreateMatrix);
529             this.groupBox7.Controls.Add(this.numericCols);
530             this.groupBox7.Controls.Add(this.numericRows);
531             this.groupBox7.Controls.Add(this.label11);
532             this.groupBox7.Controls.Add(this.label8);
533             this.groupBox7.Location = new System.Drawing.Point(418, 404)
                    ;
534             this.groupBox7.Name = "groupBox7";
535             this.groupBox7.Size = new System.Drawing.Size(104, 157);
536             this.groupBox7.TabIndex = 20;
537             this.groupBox7.TabStop = false;
538             this.groupBox7.Text = "Size";
539             //
540             // BTNCreateMatrix
541             //
542             this.BTNCreateMatrix.Location = new System.Drawing.Point(13,
                     119);
543             this.BTNCreateMatrix.Name = "BTNCreateMatrix";
544             this.BTNCreateMatrix.Size = new System.Drawing.Size(81, 23);
545             this.BTNCreateMatrix.TabIndex = 21;
546             this.BTNCreateMatrix.Text = "Create";
547             this.BTNCreateMatrix.UseVisualStyleBackColor = true;
548             this.BTNCreateMatrix.Click += new System.EventHandler(this.
                    BTNCreateMatrix_Click);
549             //
550             // numericCols
551             //
552             this.numericCols.Location = new System.Drawing.Point(13, 82)
                    ;
553             this.numericCols.Maximum = new decimal(new int[] {
554             1000,
555             0,
556             0,
557             0});
558             this.numericCols.Minimum = new decimal(new int[] {
559             10,
560             0,
561             0,
562             0});
563             this.numericCols.Name = "numericCols";
564             this.numericCols.Size = new System.Drawing.Size(81, 20);
565             this.numericCols.TabIndex = 3;
566             this.numericCols.Value = new decimal(new int[] {
567             10,
568             0,
569             0,
570             0});
571             //
```

```
572             // numericRows
573             //
574             this.numericRows.Location = new System.Drawing.Point(11, 42)
                    ;
575             this.numericRows.Maximum = new decimal(new int[] {
576             1000,
577             0,
578             0,
579             0});
580             this.numericRows.Minimum = new decimal(new int[] {
581             10,
582             0,
583             0,
584             0});
585             this.numericRows.Name = "numericRows";
586             this.numericRows.Size = new System.Drawing.Size(81, 20);
587             this.numericRows.TabIndex = 2;
588             this.numericRows.Value = new decimal(new int[] {
589             10,
590             0,
591             0,
592             0});
593             //
594             // label11
595             //
596             this.label11.AutoSize = true;
597             this.label11.Location = new System.Drawing.Point(36, 66);
598             this.label11.Name = "label11";
599             this.label11.Size = new System.Drawing.Size(27, 13);
600             this.label11.TabIndex = 1;
601             this.label11.Text = "Cols";
602             //
603             // label8
604             //
605             this.label8.AutoSize = true;
606             this.label8.Location = new System.Drawing.Point(29, 25);
607             this.label8.Name = "label8";
608             this.label8.Size = new System.Drawing.Size(34, 13);
609             this.label8.TabIndex = 0;
610             this.label8.Text = "Rows";
611             //
612             // groupBox8
613             //
614             this.groupBox8.Controls.Add(this.BTNSavePatterns);
615             this.groupBox8.Controls.Add(this.BTNSave);
616             this.groupBox8.Controls.Add(this.BTNSelectFile);
617             this.groupBox8.Location = new System.Drawing.Point(125, 557)
                    ;
618             this.groupBox8.Name = "groupBox8";
619             this.groupBox8.Size = new System.Drawing.Size(171, 75);
620             this.groupBox8.TabIndex = 21;
621             this.groupBox8.TabStop = false;
622             this.groupBox8.Text = "File options";
```

```
623             //
624             // BTNSave
625             //
626             this.BTNSave.Location = new System.Drawing.Point(90, 17);
627             this.BTNSave.Name = "BTNSave";
628             this.BTNSave.Size = new System.Drawing.Size(75, 23);
629             this.BTNSave.TabIndex = 0;
630             this.BTNSave.Text = "Save states";
631             this.BTNSave.UseVisualStyleBackColor = true;
632             this.BTNSave.Click += new System.EventHandler(this.
                    BTNSave_Click);
633             //
634             // label12
635             //
636             this.label12.AutoSize = true;
637             this.label12.Location = new System.Drawing.Point(6, 66);
638             this.label12.Name = "label12";
639             this.label12.Size = new System.Drawing.Size(83, 13);
640             this.label12.TabIndex = 22;
641             this.label12.Text = "Total population";
642             //
643             // label13
644             //
645             this.label13.AutoSize = true;
646             this.label13.Location = new System.Drawing.Point(6, 88);
647             this.label13.Name = "label13";
648             this.label13.Size = new System.Drawing.Size(47, 13);
649             this.label13.TabIndex = 23;
650             this.label13.Text = "Average";
651             //
652             // label14
653             //
654             this.label14.AutoSize = true;
655             this.label14.Location = new System.Drawing.Point(6, 110);
656             this.label14.Name = "label14";
657             this.label14.Size = new System.Drawing.Size(42, 13);
658             this.label14.TabIndex = 24;
659             this.label14.Text = "Density";
660             //
661             // groupBox9
662             //
663             this.groupBox9.Controls.Add(this.CBPoints);
664             this.groupBox9.Controls.Add(this.CheckGraphEnabled);
665             this.groupBox9.Location = new System.Drawing.Point(15, 545);
666             this.groupBox9.Name = "groupBox9";
667             this.groupBox9.Size = new System.Drawing.Size(104, 72);
668             this.groupBox9.TabIndex = 25;
669             this.groupBox9.TabStop = false;
670             this.groupBox9.Text = "Graph options";
671             //
672             // CBPoints
673             //
674             this.CBPoints.AutoSize = true;
```

```
675             this.CBPoints.Location = new System.Drawing.Point(16, 41);
676             this.CBPoints.Name = "CBPoints";
677             this.CBPoints.Size = new System.Drawing.Size(55, 17);
678             this.CBPoints.TabIndex = 1;
679             this.CBPoints.Text = "Points";
680             this.CBPoints.UseVisualStyleBackColor = true;
681             this.CBPoints.CheckedChanged += new System.EventHandler(this
                    .CBPoints_CheckedChanged);
682             //
683             // CheckGraphEnabled
684             //
685             this.CheckGraphEnabled.AutoSize = true;
686             this.CheckGraphEnabled.Checked = true;
687             this.CheckGraphEnabled.CheckState = System.Windows.Forms.
                    CheckState.Checked;
688             this.CheckGraphEnabled.Location = new System.Drawing.Point
                    (16, 23);
689             this.CheckGraphEnabled.Name = "CheckGraphEnabled";
690             this.CheckGraphEnabled.Size = new System.Drawing.Size(59,
                    17);
691             this.CheckGraphEnabled.TabIndex = 0;
692             this.CheckGraphEnabled.Text = "Enable";
693             this.CheckGraphEnabled.UseVisualStyleBackColor = true;
694             //
695             // groupBox10
696             //
697             this.groupBox10.Controls.Add(this.TXTGeneration);
698             this.groupBox10.Controls.Add(this.TXTPopulation);
699             this.groupBox10.Controls.Add(this.label14);
700             this.groupBox10.Controls.Add(this.label12);
701             this.groupBox10.Controls.Add(this.label13);
702             this.groupBox10.Location = new System.Drawing.Point(306,
                    567);
703             this.groupBox10.Name = "groupBox10";
704             this.groupBox10.Size = new System.Drawing.Size(303, 133);
705             this.groupBox10.TabIndex = 26;
706             this.groupBox10.TabStop = false;
707             this.groupBox10.Text = "Data";
708             //
709             // pictureBox10
710             //
711             this.pictureBox10.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
712             this.pictureBox10.Cursor = System.Windows.Forms.Cursors.Hand
                    ;
713             this.pictureBox10.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox10.Image")));
714             this.pictureBox10.Location = new System.Drawing.Point(581,
                    16);
715             this.pictureBox10.Name = "pictureBox10";
716             this.pictureBox10.Size = new System.Drawing.Size(38, 50);
717             this.pictureBox10.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
```

```
718             this.pictureBox10.TabIndex = 9;
719             this.pictureBox10.TabStop = false;
720             this.pictureBox10.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox10_MouseUp);
721             //
722             // pictureBox9
723             //
724             this.pictureBox9.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
725             this.pictureBox9.Cursor = System.Windows.Forms.Cursors.Hand;
726             this.pictureBox9.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox9.Image")));
727             this.pictureBox9.Location = new System.Drawing.Point(512,
                    31);
728             this.pictureBox9.Name = "pictureBox9";
729             this.pictureBox9.Size = new System.Drawing.Size(63, 26);
730             this.pictureBox9.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
731             this.pictureBox9.TabIndex = 8;
732             this.pictureBox9.TabStop = false;
733             this.pictureBox9.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox9_MouseUp);
734             //
735             // pictureBox8
736             //
737             this.pictureBox8.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
738             this.pictureBox8.Cursor = System.Windows.Forms.Cursors.Hand;
739             this.pictureBox8.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox8.Image")));
740             this.pictureBox8.Location = new System.Drawing.Point(456,
                    16);
741             this.pictureBox8.Name = "pictureBox8";
742             this.pictureBox8.Size = new System.Drawing.Size(50, 50);
743             this.pictureBox8.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
744             this.pictureBox8.TabIndex = 7;
745             this.pictureBox8.TabStop = false;
746             this.pictureBox8.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox8_MouseUp);
747             //
748             // pictureBox7
749             //
750             this.pictureBox7.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
751             this.pictureBox7.Cursor = System.Windows.Forms.Cursors.Hand;
752             this.pictureBox7.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox7.Image")));
753             this.pictureBox7.Location = new System.Drawing.Point(412,
                    16);
754             this.pictureBox7.Name = "pictureBox7";
755             this.pictureBox7.Size = new System.Drawing.Size(38, 50);
756             this.pictureBox7.SizeMode = System.Windows.Forms.
```

```
            PictureBoxSizeMode.StretchImage;
757         this.pictureBox7.TabIndex = 6;
758         this.pictureBox7.TabStop = false;
759         this.pictureBox7.MouseUp += new System.Windows.Forms.
                MouseEventHandler(this.pictureBox7_MouseUp);
760         //
761         // pictureBox6
762         //
763         this.pictureBox6.BorderStyle = System.Windows.Forms.
                BorderStyle.FixedSingle;
764         this.pictureBox6.Cursor = System.Windows.Forms.Cursors.Hand;
765         this.pictureBox6.Image = ((System.Drawing.Image)(resources.
                GetObject("pictureBox6.Image")));
766         this.pictureBox6.Location = new System.Drawing.Point(329,
                16);
767         this.pictureBox6.Name = "pictureBox6";
768         this.pictureBox6.Size = new System.Drawing.Size(77, 50);
769         this.pictureBox6.SizeMode = System.Windows.Forms.
                PictureBoxSizeMode.StretchImage;
770         this.pictureBox6.TabIndex = 5;
771         this.pictureBox6.TabStop = false;
772         this.pictureBox6.MouseUp += new System.Windows.Forms.
                MouseEventHandler(this.pictureBox6_MouseUp);
773         //
774         // pictureBox5
775         //
776         this.pictureBox5.BorderStyle = System.Windows.Forms.
                BorderStyle.FixedSingle;
777         this.pictureBox5.Cursor = System.Windows.Forms.Cursors.Hand;
778         this.pictureBox5.Image = ((System.Drawing.Image)(resources.
                GetObject("pictureBox5.Image")));
779         this.pictureBox5.Location = new System.Drawing.Point(273,
                16);
780         this.pictureBox5.Name = "pictureBox5";
781         this.pictureBox5.Size = new System.Drawing.Size(50, 50);
782         this.pictureBox5.SizeMode = System.Windows.Forms.
                PictureBoxSizeMode.StretchImage;
783         this.pictureBox5.TabIndex = 4;
784         this.pictureBox5.TabStop = false;
785         this.pictureBox5.MouseUp += new System.Windows.Forms.
                MouseEventHandler(this.pictureBox5_MouseUp);
786         //
787         // pictureBox4
788         //
789         this.pictureBox4.BorderStyle = System.Windows.Forms.
                BorderStyle.FixedSingle;
790         this.pictureBox4.Cursor = System.Windows.Forms.Cursors.Hand;
791         this.pictureBox4.Image = ((System.Drawing.Image)(resources.
                GetObject("pictureBox4.Image")));
792         this.pictureBox4.Location = new System.Drawing.Point(217,
                16);
793         this.pictureBox4.Name = "pictureBox4";
794         this.pictureBox4.Size = new System.Drawing.Size(50, 50);
```

```
795             this.pictureBox4.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
796             this.pictureBox4.TabIndex = 3;
797             this.pictureBox4.TabStop = false;
798             this.pictureBox4.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox4_MouseUp);
799             //
800             // pictureBox3
801             //
802             this.pictureBox3.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
803             this.pictureBox3.Cursor = System.Windows.Forms.Cursors.Hand;
804             this.pictureBox3.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox3.Image")));
805             this.pictureBox3.Location = new System.Drawing.Point(161,
                    16);
806             this.pictureBox3.Name = "pictureBox3";
807             this.pictureBox3.Size = new System.Drawing.Size(50, 50);
808             this.pictureBox3.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
809             this.pictureBox3.TabIndex = 2;
810             this.pictureBox3.TabStop = false;
811             this.pictureBox3.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox3_MouseUp);
812             //
813             // pictureBox2
814             //
815             this.pictureBox2.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
816             this.pictureBox2.Cursor = System.Windows.Forms.Cursors.Hand;
817             this.pictureBox2.ErrorImage = ((System.Drawing.Image)(
                    resources.GetObject("pictureBox2.ErrorImage")));
818             this.pictureBox2.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox2.Image")));
819             this.pictureBox2.Location = new System.Drawing.Point(70, 16)
                    ;
820             this.pictureBox2.Name = "pictureBox2";
821             this.pictureBox2.Size = new System.Drawing.Size(85, 50);
822             this.pictureBox2.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
823             this.pictureBox2.TabIndex = 1;
824             this.pictureBox2.TabStop = false;
825             this.pictureBox2.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox2_MouseUp);
826             //
827             // pictureBox1
828             //
829             this.pictureBox1.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
830             this.pictureBox1.Cursor = System.Windows.Forms.Cursors.Hand;
831             this.pictureBox1.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox1.Image")));
832             this.pictureBox1.Location = new System.Drawing.Point(14, 16)
```

```
            ;
833         this.pictureBox1.Name = "pictureBox1";
834         this.pictureBox1.Size = new System.Drawing.Size(50, 50);
835         this.pictureBox1.SizeMode = System.Windows.Forms.
                PictureBoxSizeMode.StretchImage;
836         this.pictureBox1.TabIndex = 0;
837         this.pictureBox1.TabStop = false;
838         this.pictureBox1.MouseUp += new System.Windows.Forms.
                MouseEventHandler(this.pictureBox1_MouseUp);
839         //
840         // groupBox11
841         //
842         this.groupBox11.Controls.Add(this.pictureBox1);
843         this.groupBox11.Controls.Add(this.pictureBox2);
844         this.groupBox11.Controls.Add(this.pictureBox10);
845         this.groupBox11.Controls.Add(this.pictureBox3);
846         this.groupBox11.Controls.Add(this.pictureBox5);
847         this.groupBox11.Controls.Add(this.pictureBox4);
848         this.groupBox11.Controls.Add(this.pictureBox9);
849         this.groupBox11.Controls.Add(this.pictureBox6);
850         this.groupBox11.Controls.Add(this.pictureBox7);
851         this.groupBox11.Controls.Add(this.pictureBox8);
852         this.groupBox11.Location = new System.Drawing.Point(621, 23)
                ;
853         this.groupBox11.Name = "groupBox11";
854         this.groupBox11.Size = new System.Drawing.Size(633, 74);
855         this.groupBox11.TabIndex = 27;
856         this.groupBox11.TabStop = false;
857         this.groupBox11.Text = "Patterns";
858         //
859         // menuStrip1
860         //
861         this.menuStrip1.Items.AddRange(new System.Windows.Forms.
                ToolStripItem[] {
862         this.generarPatronesToolStripMenuItem});
863         this.menuStrip1.Location = new System.Drawing.Point(0, 0);
864         this.menuStrip1.Name = "menuStrip1";
865         this.menuStrip1.Size = new System.Drawing.Size(1266, 24);
866         this.menuStrip1.TabIndex = 28;
867         this.menuStrip1.Text = "menuStrip1";
868         //
869         // generarPatronesToolStripMenuItem
870         //
871         this.generarPatronesToolStripMenuItem.DropDownItems.AddRange
                (new System.Windows.Forms.ToolStripItem[] {
872         this.generatePatternsToolStripMenuItem});
873         this.generarPatronesToolStripMenuItem.Name = "
                generarPatronesToolStripMenuItem";
874         this.generarPatronesToolStripMenuItem.Size = new System.
                Drawing.Size(37, 20);
875         this.generarPatronesToolStripMenuItem.Text = "File";
876         //
877         // generatePatternsToolStripMenuItem
```

```
878             //
879             this.generatePatternsToolStripMenuItem.Name = "
                    generatePatternsToolStripMenuItem";
880             this.generatePatternsToolStripMenuItem.Size = new System.
                    Drawing.Size(167, 22);
881             this.generatePatternsToolStripMenuItem.Text = "Generate
                    patterns";
882             this.generatePatternsToolStripMenuItem.Click += new System.
                    EventHandler(this.
                    generatePatternsToolStripMenuItem_Click);
883             //
884             // BTNSavePatterns
885             //
886             this.BTNSavePatterns.Location = new System.Drawing.Point(36,
                     46);
887             this.BTNSavePatterns.Name = "BTNSavePatterns";
888             this.BTNSavePatterns.Size = new System.Drawing.Size(96, 23);
889             this.BTNSavePatterns.TabIndex = 19;
890             this.BTNSavePatterns.Text = "Save patterns";
891             this.BTNSavePatterns.UseVisualStyleBackColor = true;
892             this.BTNSavePatterns.Click += new System.EventHandler(this.
                    BTNSavePatterns_Click);
893             //
894             // CBPatternRecognition
895             //
896             this.CBPatternRecognition.AutoSize = true;
897             this.CBPatternRecognition.Location = new System.Drawing.
                    Point(24, 48);
898             this.CBPatternRecognition.Name = "CBPatternRecognition";
899             this.CBPatternRecognition.Size = new System.Drawing.Size
                    (115, 17);
900             this.CBPatternRecognition.TabIndex = 29;
901             this.CBPatternRecognition.Text = "Pattern recognition";
902             this.CBPatternRecognition.UseVisualStyleBackColor = true;
903             //
904             // Form1
905             //
906             this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F)
                    ;
907             this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
                    ;
908             this.ClientSize = new System.Drawing.Size(1266, 710);
909             this.Controls.Add(this.groupBox11);
910             this.Controls.Add(this.groupBox10);
911             this.Controls.Add(this.groupBox9);
912             this.Controls.Add(this.groupBox8);
913             this.Controls.Add(this.groupBox7);
914             this.Controls.Add(this.BTNClear);
915             this.Controls.Add(this.groupBox6);
916             this.Controls.Add(this.groupBox5);
917             this.Controls.Add(this.groupBox4);
918             this.Controls.Add(this.BTNZoomM);
919             this.Controls.Add(this.BTNZoomP);
```

```
920             this.Controls.Add(this.groupBox3);
921             this.Controls.Add(this.groupBox2);
922             this.Controls.Add(this.groupBox1);
923             this.Controls.Add(this.flowLayoutPanel1);
924             this.Controls.Add(this.menuStrip1);
925             this.MainMenuStrip = this.menuStrip1;
926             this.Name = "Form1";
927             this.Text = "  ";
928             ((System.ComponentModel.ISupportInitialize)(this.
                    PBAutomataSimulator)).EndInit();
929             ((System.ComponentModel.ISupportInitialize)(this.CHHistogram
                    )).EndInit();
930             this.groupBox1.ResumeLayout(false);
931             this.groupBox2.ResumeLayout(false);
932             this.groupBox2.PerformLayout();
933             ((System.ComponentModel.ISupportInitialize)(this.TBSpeed)).
                    EndInit();
934             this.flowLayoutPanel1.ResumeLayout(false);
935             this.flowLayoutPanel1.PerformLayout();
936             this.groupBox3.ResumeLayout(false);
937             this.groupBox3.PerformLayout();
938             this.groupBox4.ResumeLayout(false);
939             this.groupBox4.PerformLayout();
940             this.groupBox5.ResumeLayout(false);
941             this.groupBox5.PerformLayout();
942             ((System.ComponentModel.ISupportInitialize)(this.numericOnes
                    )).EndInit();
943             this.groupBox6.ResumeLayout(false);
944             this.groupBox7.ResumeLayout(false);
945             this.groupBox7.PerformLayout();
946             ((System.ComponentModel.ISupportInitialize)(this.numericCols
                    )).EndInit();
947             ((System.ComponentModel.ISupportInitialize)(this.numericRows
                    )).EndInit();
948             this.groupBox8.ResumeLayout(false);
949             this.groupBox9.ResumeLayout(false);
950             this.groupBox9.PerformLayout();
951             this.groupBox10.ResumeLayout(false);
952             this.groupBox10.PerformLayout();
953             ((System.ComponentModel.ISupportInitialize)(this.
                    pictureBox10)).EndInit();
954             ((System.ComponentModel.ISupportInitialize)(this.pictureBox9
                    )).EndInit();
955             ((System.ComponentModel.ISupportInitialize)(this.pictureBox8
                    )).EndInit();
956             ((System.ComponentModel.ISupportInitialize)(this.pictureBox7
                    )).EndInit();
957             ((System.ComponentModel.ISupportInitialize)(this.pictureBox6
                    )).EndInit();
958             ((System.ComponentModel.ISupportInitialize)(this.pictureBox5
                    )).EndInit();
959             ((System.ComponentModel.ISupportInitialize)(this.pictureBox4
                    )).EndInit();
```

```
960             ((System.ComponentModel.ISupportInitialize)(this.pictureBox3
                    )).EndInit();
961             ((System.ComponentModel.ISupportInitialize)(this.pictureBox2
                    )).EndInit();
962             ((System.ComponentModel.ISupportInitialize)(this.pictureBox1
                    )).EndInit();
963             this.groupBox11.ResumeLayout(false);
964             this.menuStrip1.ResumeLayout(false);
965             this.menuStrip1.PerformLayout();
966             this.ResumeLayout(false);
967             this.PerformLayout();
968
969         }
970
971         #endregion
972
973         private System.Windows.Forms.PictureBox PBAutomataSimulator;
974         private System.Windows.Forms.DataVisualization.Charting.Chart
                CHHistogram;
975         private System.Windows.Forms.GroupBox groupBox1;
976         private System.Windows.Forms.Button BTNStart;
977         private System.Windows.Forms.GroupBox groupBox2;
978         private System.Windows.Forms.Label TXTGeneration;
979         private System.Windows.Forms.Label label5;
980         private System.Windows.Forms.Label label4;
981         private System.Windows.Forms.Label label3;
982         private System.Windows.Forms.Label label2;
983         private System.Windows.Forms.ComboBox ComboBY2i;
984         private System.Windows.Forms.ComboBox ComboBX2i;
985         private System.Windows.Forms.ComboBox ComboBYi;
986         private System.Windows.Forms.ComboBox ComboBXi;
987         private System.Windows.Forms.Label TXTPopulation;
988         private System.Windows.Forms.Button BTNStep;
989         private System.Windows.Forms.TrackBar TBSpeed;
990         private System.Windows.Forms.Timer TimerSimulation;
991         private System.Windows.Forms.GroupBox groupBox3;
992         private System.Windows.Forms.Label label6;
993         private System.Windows.Forms.Label label1;
994         private System.Windows.Forms.Button BTNZoomP;
995         private System.Windows.Forms.Button BTNZoomM;
996         private System.Windows.Forms.GroupBox groupBox4;
997         private System.Windows.Forms.GroupBox groupBox5;
998         private System.Windows.Forms.Button button1;
999         private System.Windows.Forms.NumericUpDown numericOnes;
1000        private System.Windows.Forms.Label label7;
1001        private System.Windows.Forms.GroupBox groupBox6;
1002        private System.Windows.Forms.Button BTNSelectFile;
1003        private System.Windows.Forms.Button BTNClear;
1004        private System.Windows.Forms.GroupBox groupBox7;
1005        private System.Windows.Forms.NumericUpDown numericCols;
1006        private System.Windows.Forms.NumericUpDown numericRows;
1007        private System.Windows.Forms.Label label11;
1008        private System.Windows.Forms.Label label8;
```

```
1009            private System.Windows.Forms.Button BTNCreateMatrix;
1010            private System.Windows.Forms.GroupBox groupBox8;
1011            private System.Windows.Forms.Button BTNSave;
1012            private System.Windows.Forms.Label label12;
1013            private System.Windows.Forms.Label label13;
1014            private System.Windows.Forms.Label label14;
1015            private System.Windows.Forms.GroupBox groupBox9;
1016            private System.Windows.Forms.CheckBox CheckGraphEnabled;
1017            private System.Windows.Forms.GroupBox groupBox10;
1018            public System.Windows.Forms.FlowLayoutPanel flowLayoutPanel1;
1019            private System.Windows.Forms.PictureBox pictureBox1;
1020            private System.Windows.Forms.PictureBox pictureBox10;
1021            private System.Windows.Forms.PictureBox pictureBox9;
1022            private System.Windows.Forms.PictureBox pictureBox8;
1023            private System.Windows.Forms.PictureBox pictureBox7;
1024            private System.Windows.Forms.PictureBox pictureBox6;
1025            private System.Windows.Forms.PictureBox pictureBox5;
1026            private System.Windows.Forms.PictureBox pictureBox4;
1027            private System.Windows.Forms.PictureBox pictureBox3;
1028            private System.Windows.Forms.PictureBox pictureBox2;
1029            private System.Windows.Forms.GroupBox groupBox11;
1030            private System.Windows.Forms.ColorDialog colorDialog;
1031            private System.Windows.Forms.Button BTNGrid;
1032            private System.Windows.Forms.Button BTNDeadCells;
1033            private System.Windows.Forms.Button BTNAliveCells;
1034            private System.Windows.Forms.CheckBox CBPoints;
1035            private System.Windows.Forms.MenuStrip menuStrip1;
1036            private System.Windows.Forms.ToolStripMenuItem
                    generarPatronesToolStripMenuItem;
1037            private System.Windows.Forms.ToolStripMenuItem
                    generatePatternsToolStripMenuItem;
1038            private System.Windows.Forms.Button BTNSavePatterns;
1039            private System.Windows.Forms.CheckBox CBPatternRecognition;
1040        }
1041    }
```

Posteriormente tenemos la clase que contiene el código que hace que el autómata se comporte como fue indicado.

```
 1  using System;
 2  using System.Drawing;
 3  using System.Windows.Forms;
 4  using System.IO;
 5  using System.Collections;
 6  using System.Collections.Generic;
 7  using System.Linq;
 8  using System.Threading;
 9
10  namespace GameOfLife
11  {
12
13      public partial class Form1 : Form
14      {
15
```

```
16          /*************************************
17           *           GLOBAL VARIABLES          *
18           *************************************/
19
20          private uint[,] matrix;
21
22          private int cellArea = 10;
23          private long acumOnes = 0;
24
25          private int generation = 1;
26          private int total_cells = 0;
27
28          private Brush alive = Brushes.White;
29          private Brush dead = Brushes.Black;
30          private Pen grid = Pens.Gray;
31
32          private bool move;
33          private uint DEAD = 0;
34          private uint ALIVE = 16777215;
35
36          private String[] colors = { "White", "Black", "Red", "Blue", "
                Green", "Yellow", "Violet", "Gray" };
37
38          /*************************************
39           *          PATTERN RECOGNITION        *
40           *************************************/
41          //Here we gonna store all the paterns.
42          //Dictionary is an element that works as a hash table, so the
                first element
43          //it's the key the second element it's the value, and for
                convinence we selected
44          //a tuple as the value.
45          //A tuple it's an equivalent of pair in C++, and we can get each
                 element using
46          //Tuple.Item1 and Tuple.Item2. For our case the first item will
                contain
47          //the name or key of a finite automata and the second element
                will contain
48          //the next "state" of the current automata
49          private Dictionary<ulong, ulong> data = new Dictionary<ulong,
                ulong>();
50          private Dictionary<ulong, List<ulong>> recurrences = new
                Dictionary<ulong, List<ulong>>();
51          private Dictionary<string, Graph> clasifications = new
                Dictionary<string, Graph>();
52          //Here are all the patterns that we can generate, from 2x2 to 4
                x4
53          private List<Dictionary<ulong, ulong>> patterns = new List<
                Dictionary<ulong, ulong>>();
54          //To make more efficient this application we gonna use threads
55          Thread[] thread = new Thread[6];
56
57          private List<uint[,]> figure = new List<uint[,]>();
```

```
58          private int index_pattern = 0;
59
60          /// <summary>
61          /// Constructor
62          /// </summary>
63          public Form1()
64          {
65              //Creating UI elements
66              InitializeComponent();
67              //We init all the predefined figures
68              initMosaics();
69              //Init the program with a 100, 100 matrix
70              createMatrix(100, 100);
71              //Make a responsive GUI
72              scrollBox();
73              //Creating each dictionary for our patterns
74              for (int i = 0; i < 6; i++) {
75                  patterns.Add(new Dictionary<ulong, ulong>());
76              }
77          }
78
79          private Color getColor() {
80              DialogResult result = colorDialog.ShowDialog();
81              return colorDialog.Color;
82          }
83          /// <summary>
84          /// This function creates a matrix of bools which size it's n x
                m
85          /// also this function adds an extra pair of cols and rows to
86          /// simulate a toroid
87          /// </summary>
88          /// <param name="rows"></param>
89          /// <param name="cols"></param>
90          private void createMatrix(int rows, int cols)
91          {
92              matrix = new uint[cols, rows];
93              scrollBox();
94          }
95
96          /// <summary>
97          /// This method paints the matrix in the Paint Box
98          /// </summary>
99          /// <param name="sender"></param>
100         /// <param name="e"></param>
101         private void PBAutomataSimulator_Paint(object sender,
                PaintEventArgs e)
102         {
103             int x_size = matrix.GetLength(0);
104             int y_size = matrix.GetLength(1);
105             total_cells = x_size * y_size;
106
107             Graphics graphics = e.Graphics;
108
```

```
109             for (int row = 0; row < x_size; row++)
110             {
111
112                 for (int col = 0; col < y_size; col++)
113                 {
114
115                     if (matrix[row, col] != DEAD)
116                     {
117                         SolidBrush aliveCellColor = new SolidBrush(
                                ColorHandler.fromIntToGradient(matrix[row,
                                col], ALIVE));
118                         graphics.FillRectangle(aliveCellColor, row *
                                cellArea, col * cellArea, cellArea, cellArea
                                );
119                     }
120                     else
121                         graphics.FillRectangle(dead, row * cellArea, col
                                 * cellArea, cellArea, cellArea);
122                 }
123             }
124
125             for (int y = 0; y < y_size; y++)
126             {
127                 graphics.DrawLine(grid, 0, y * cellArea, total_cells *
                        cellArea, y * cellArea);
128             }
129
130             for (int x = 0; x < x_size; x++)
131             {
132                 graphics.DrawLine(grid, x * cellArea, 0, x * cellArea,
                        total_cells * cellArea);
133             }
134         }
135
136         /// <summary>
137         /// This funcion manipulates a matrix and evaluate it
138         /// using our rules.
139         /// </summary>
140         /// <param name="p_matrix"></param>
141         /// <returns>A matrix with the new generation data</returns>
142         private uint[,] nextGeneration(uint[,] p_matrix)
143         {
144
145             /*****************************************************
146              *                      CONDITIONS                   *
147              * ***************************************************/
148             //23 33 GAME OF LIFE
149             //77 22 DIFFUSION
150             /****************** X values ********************/
151             int Xi = Int32.Parse(string.IsNullOrEmpty(ComboBXi.Text) ? "
                    2" : ComboBXi.Text);
152
153             /****************** Y values ********************/
```

```
154             int Yi = Int32.Parse(string.IsNullOrEmpty(ComboBYi.Text) ? "
                    3" : ComboBYi.Text);
155
156             /****************** X2 values ******************/
157             int X2i = Int32.Parse((string.IsNullOrEmpty(ComboBX2i.Text)
                    ? "3" : ComboBX2i.Text));
158
159             /****************** Y2 values ******************/
160             int Y2i = Int32.Parse((string.IsNullOrEmpty(ComboBY2i.Text)
                    ? "3" : ComboBY2i.Text));
161
162             uint[,] new_matrix = new uint[p_matrix.GetLength(0),
                    p_matrix.GetLength(1)];
163             //We check each cell from the original matrix and we
                    substitute it
164             for (int row = 0; row < p_matrix.GetLength(0); row++)
165             {
166
167                 for (int col = 0; col < p_matrix.GetLength(1); col++)
168                 {
169                     /**
170                      * Here we need to evaluate using the rules given by
                             input
171                      **/
172                     int neighbors = getAliveNeighbors(p_matrix, row, col
                            );
173                     //If the cell is alive
174                     if (p_matrix[row, col] != DEAD)
175                     {
176                         uint color_cell = p_matrix[row, col] - 1;
177                         new_matrix[row, col] = (neighbors >= Xi &&
                                neighbors <= Yi) ? (color_cell) : DEAD;
178                     }
179                     //If the central cell is dead
180                     else
181                     {
182                         new_matrix[row, col] = (neighbors >= X2i &&
                                neighbors <= Y2i) ? ALIVE : DEAD;
183                     }
184                 }
185
186             }
187
188             return new_matrix;
189         }
190
191         /// <summary>
192         /// Gets information about the cells around a central cell.
193         /// Obviouslly the cells must to be alive.
194         /// </summary>
195         /// <param name="p_matrix">The actual matrix</param>
196         /// <param name="p_row">row of the central cell</param>
197         /// <param name="p_col">col of the central cell</param>
```

```
198         /// <returns>Number of neighboors around the central cell (just
                living neighbors)</returns>
199         private int getAliveNeighbors(uint[,] p_matrix, int p_row, int
                p_col)
200         {
201
202             int neighbors = 0;
203             int max_x = p_matrix.GetLength(0);
204             int max_y = p_matrix.GetLength(1);
205             uint[,] sub_matrix = new uint[3, 3];
206
207             for (int row = -1, sx = 0; row <= 1; row++, sx++)
208             {
209
210                 for (int col = -1, sy = 0; col <= 1; col++, sy++)
211                 {
212
213                     int x = row + p_row;
214                     int y = col + p_col;
215
216                     //We are in the center cell
217                     if (x == p_row && y == p_col)
218                     {
219                         sub_matrix[sx, sy] = p_matrix[x, y];
220                         continue;
221                     }
222                     // ----- Corners ----- //
223
224                     //Up-Left
225                     if (x == -1 && y == -1 && (p_matrix[max_x - 1, max_y
                            - 1] != DEAD))
226                     {
227                         sub_matrix[sx, sy] = p_matrix[max_x - 1, max_y -
                                1];
228                         neighbors++;
229                     }
230                     //Down-Right
231                     if (x == max_x && y == max_y && (p_matrix[0, 0] !=
                            DEAD))
232                     {
233                         sub_matrix[sx, sy] = p_matrix[0, 0];
234                         neighbors++;
235                     }
236                     //Up-Right
237                     if (x == -1 && y == max_y && (p_matrix[max_x - 1, 0]
                            != DEAD))
238                     {
239                         sub_matrix[sx, sy] = p_matrix[max_x - 1, 0];
240                         neighbors++;
241                     }
242                     //Down-left
243                     if (x == max_x && y == -1 && (p_matrix[0, max_y - 1]
                            != DEAD))
```

```
244                     {
245                         sub_matrix[sx, sy] = p_matrix[0, max_y - 1];
246                         neighbors++;
247                     }
248                     // ------ Edges ------ //
249 
250                     if (y >= 0 && y < max_y)
251                     {
252                         //Up
253                         if (x == -1 && p_matrix[max_x - 1, y] != DEAD)
254                         {
255                             sub_matrix[sx, sy] = p_matrix[max_x - 1, y];
256                             neighbors++;
257                         }
258                         //Down
259                         else if (x == max_x && p_matrix[0, y] != DEAD)
260                         {
261                             sub_matrix[sx, sy] = p_matrix[0, y];
262                             neighbors++;
263                         }
264                     }
265                     if (x >= 0 && x < max_x)
266                     {
267                         //Right
268                         if (y == -1 && p_matrix[x, max_y - 1] != DEAD)
269                         {
270                             sub_matrix[sx, sy] = p_matrix[x, max_y - 1];
271                             neighbors++;
272                         }
273                         //Left
274                         else if (y == max_y && p_matrix[x, 0] != DEAD)
275                         {
276                             sub_matrix[sx, sy] = p_matrix[x, 0];
277                             neighbors++;
278                         }
279                     }
280 
281                     if (x < 0 || x >= max_x)
282                     {
283                         continue;
284                     }
285 
286                     if (y < 0 || y >= max_y)
287                     {
288                         continue;
289                     }
290 
291                     if (p_matrix[x, y] != DEAD)
292                     {
293                         sub_matrix[sx, sy] = p_matrix[x, y];
294                         neighbors++;
295                     }
296 
```

```
297                 }
298             }
299             if (CBPatternRecognition.Checked) {
300                 patternRecognition(generateMatrixPatterns(p_matrix,
                        p_row, p_col, 2));
301                 patternRecognition(sub_matrix);
302                 patternRecognition(generateMatrixPatterns(p_matrix,
                        p_row, p_col, 4));
303             }
304             return neighbors;
305         }
306 
307         /// <summary>
308         /// This method calls nextGeneration method and
309         /// updates the GUI and the count of our alive cells
310         /// </summary>
311         private void step()
312         {
313             matrix = nextGeneration(matrix);
314             updateTextGeneration();
315             countOnes();
316             PBAutomataSimulator.Invalidate();
317         }
318 
319         /// <summary>
320         /// Here we just change the text that show us
321         /// the number of generations
322         /// </summary>
323         private void updateTextGeneration()
324         {
325             TXTGeneration.Text = "Generation: " + generation++;
326         }
327 
328         /// <summary>
329         /// This method make a rezise of the Paint Box and flow layout
                panel
330         /// it makes possible make zoom and the movement into the GUI
331         /// </summary>
332         private void scrollBox()
333         {
334             PBAutomataSimulator.Size = new Size((matrix.GetLength(0)) *
                    cellArea, (matrix.GetLength(1)) * cellArea);
335             PBAutomataSimulator.SizeMode = PictureBoxSizeMode.AutoSize;
336             flowLayoutPanel1.AutoScroll = true;
337             flowLayoutPanel1.Controls.Add(PBAutomataSimulator);
338         }
339 
340         /// <summary>
341         /// As you can imagine here we just get the number of ones
342         /// in our matrix (alive cells)
343         /// </summary>
344         private void countOnes()
345         {
```

```
346             int ones = 0;
347             for (int x = 0; x < matrix.GetLength(0); x++)
348             {
349 
350                 for (int y = 0; y < matrix.GetLength(1); y++)
351                 {
352                     if (matrix[x, y] != DEAD) ones++;
353                 }
354             }
355 
356             if (CheckGraphEnabled.Checked)
357                 CHHistogram.Series["#Ones"].Points.AddY(ones);
358             TXTPopulation.Text = "Population " + ones;
359             acumOnes += ones;
360             double val = acumOnes / generation;
361             label12.Text = "Total Population: " + acumOnes;
362             label13.Text = "Average: " + (val);
363             label14.Text = "Density: " + (val / (matrix.GetLength(0) *
                    matrix.GetLength(1)));
364         }
365 
366         /*
                ************************************************************************************
367          *                                          SECOND TERM CODE                                *
368          ************************************************************************************
                 */
369 
370         /// <summary>
371         /// This funciton generates multiple binary string one per each
372         /// matrix dimension. For our case the maximum matrix will be
373         /// of 7 x 7 at most
374         /// </summary>
375         private void generatePatterns(int size)
376         {
377             data.Clear();
378             recurrences.Clear();
379             clasifications.Clear();
380             Console.WriteLine("generatePatterns(" + size + ")");
381             //To generate all the possible combinations inside
382             //a matrix from 2x2 to 8x8 (just square) we gonna
383             //to convert from a decimal number to a binary string
384             //so, size_string contains the limit of combinations
385             //in each matrix n^2 where n is the size of each matrix
386             try
387             {
388                 int size_string = 0;
389                 ulong n_combinations = 0;
390                 SaveFileDialog saveFileDialog = new SaveFileDialog();
391                 saveFileDialog.Filter = "Archivo de texto|*.txt";
392                 saveFileDialog.Title = "Patterns file name";
393                 saveFileDialog.ShowDialog();
```

```
394                 StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
395                 //Size of the binary string: 2^2, 3^2, 4^2, ... , 7^2
396                 size_string = size * size;
397                 //Now we get the number of combinations it is 2^
                        size_string
398                 n_combinations = (ulong)Math.Pow(2.0, size_string);
399                 sw.Write("GraphPlot[{");
400                 //We iterate from 0 to 2^n and convert this number to a
                        binary string
401                 for (ulong j = 1; j < n_combinations; j++)
402                 {
403                     //We convert j to a binary string
404                     string str_binary = Convert.ToString((long)j, 2);
405                     while (str_binary.Length != size_string)
406                     {
407                         str_binary = "0" + str_binary;
408                     }
409                     uint[,] next_state = nextGeneration(
                            fromBinaryToMatrix(str_binary));
410                     string str_next_state = fromMatrixToString(
                            next_state);
411                     ulong nextState = Convert.ToUInt64(str_next_state,
                            2);
412                     if (saveFileDialog != null) {
413                         sw.Write(j + "->" + nextState + ((j <
                                n_combinations - 1) ? ", " : ""));
414                         data.Add(j, nextState);
415                     }
416                 }
417                 sw.Write("}]");
418                 sw.Close();
419                 MessageBox.Show("I've stored something");
420                 //Sorting the paterns created
421                 sortTransitions();
422                 //Create some objects with the paterns
423                 CreateGraphObjects();
424                 //We create a file to send it to mathematica
425                 outputMathematica();
426             }
427             catch (Exception e) {
428                 Console.WriteLine("An exception has occurred on
                        generatePatterns " + e);
429             }
430         }
431
432         /// <summary>
433         /// Generate a txt file with the structure of a mathematica file
434         /// </summary>
435         private void outputMathematica() {
436             Dictionary<ulong, ulong> unique_nodes = new Dictionary<ulong
                    , ulong>();
437             try {
```

```
438                 SaveFileDialog saveFileDialog = new SaveFileDialog();
439                 saveFileDialog.Filter = "Archivo de texto|*.txt";
440                 saveFileDialog.Title = "Paterns filtered";
441                 saveFileDialog.ShowDialog();
442                 StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
443
444                 sw.Write("GraphPlot[{");
445                 foreach (KeyValuePair<string, Graph> item_graph in
                        clasifications) {
446                     Graph current_graph = item_graph.Value;
447                     Dictionary<ulong, List<ulong>> node = current_graph.
                            getAllNodes();
448                     foreach (KeyValuePair<ulong, List<ulong>> item in
                            node) {
449
450                         for (int i = 0; i < item.Value.Count; i++) {
451                             if (!unique_nodes.ContainsKey(item.Value[i])
                                    )
452                                 unique_nodes.Add(item.Value[i], item.Key
                                        );
453                         }
454                     }
455                 }
456
457                 foreach (KeyValuePair<ulong, ulong> item in unique_nodes
                        )
458                     sw.Write(item.Key + "->" + item.Value + ((item.Key
                            == unique_nodes.Last().Key) ? "" : ","));
459                 sw.Write("}]");
460                 sw.Close();
461                 MessageBox.Show("I've stored something  ");
462             }
463             catch (Exception e) {
464                 Console.WriteLine("An exception has occurred creating a
                        mathematica file " + e);
465             }
466         }
467         /// <summary>
468         /// This function generate a matrix using a binary string.
469         /// We are just considering a square matrix, so we can
470         /// calculate the number of rows and cols calculating
471         /// the square root of the binary_string size
472         /// </summary>
473         /// <param name="binary_string">This string
474         /// contains a matrix but in a dimension</param>
475         private uint[,] fromBinaryToMatrix(string binary_string)
476         {
477             //We get the size of our sub_matrix. As we know
478             //the matrix it's a square, so we need to calculate
479             //the square root of the length of the binary string
480             int size = Convert.ToInt16(Math.Sqrt(binary_string.Length));
481             //We create a new boolean matrix
```

```
482             uint[,] sub_matrix = new uint[size, size];
483             //We build the sub matrix using our binary string
484             for (int x = 0, position = 0; x < size; x++)
485             {
486                 for (int y = 0; y < size; y++)
487                 {
488                     //We add an element int the x'th row in the y'th
                        position
489                     //If this element it's equals to an one we put true
                        in out sub matrix
490                     sub_matrix[x, y] = (binary_string[position++] == '0'
                        ) ? DEAD : ALIVE;
491 
492                 }
493 
494             }
495             return sub_matrix;
496         }
497 
498         /// <summary>
499         /// Convert a matrix to a binary string
500         /// </summary>
501         /// <param name="a_matrix">Source matrix</param>
502         /// <returns>Binary string</returns>
503         private string fromMatrixToString(uint[,] a_matrix)
504         {
505             string str = "";
506 
507             for (int i = 0; i < a_matrix.GetLength(0); i++) {
508                 for (int j = 0; j < a_matrix.GetLength(1); j++) {
509                     str += (a_matrix[i, j] == DEAD ? "0" : "1");
510                 }
511             }
512 
513             return str;
514         }
515 
516         /// <summary>
517         /// Just for testing
518         /// </summary>
519         /// <param name="a_matrix">Source matrix</param>
520         private void printMatrix(int[,] a_matrix) {
521 
522             Console.WriteLine("
                    ---------------------------------------------------- ");
523             for (int i = 0; i < a_matrix.GetLength(0); i++) {
524 
525                 for (int j = 0; j < a_matrix.GetLength(1); j++) {
526                     Console.Write(" " + ((a_matrix[i, j] == DEAD) ? "1"
                        : "0"));
527                 }
528                 Console.WriteLine();
529             }
```

```
530             Console.WriteLine("
                    ------------------------------------------------------ ");
531         }
532         /// <summary>
533         /// This function
534         /// </summary>
535         /// <param name="sub_matrix"></param>
536         private void patternRecognition(uint[,] p_matrix)
537         {
538             int dimension = p_matrix.GetLength(0);
539             ulong key = Convert.ToUInt64(fromMatrixToString(p_matrix),
                    2);
540 
541             if (!patterns[dimension].ContainsKey(key))
542                 patterns[dimension].Add(key, 1);
543             else
544                 patterns[dimension][key]++;
545         }
546 
547         private uint[,] generateMatrixPatterns(uint[,] p_matrix, int
                p_row, int p_col, int dimension) {
548             int init = -1;
549             int end = 0;
550 
551             if (dimension == 4)
552             {
553                 init = -2;
554                 end = 1;
555             }
556 
557             int max_x = p_matrix.GetLength(0);
558             int max_y = p_matrix.GetLength(1);
559             uint[,] sub_matrix = new uint[dimension, dimension];
560 
561             for (int row = init, sx = 0; row <= end; row++, sx++)
562             {
563 
564                 for (int col = init, sy = 0; col <= end; col++, sy++)
565                 {
566 
567                     int x = row + p_row;
568                     int y = col + p_col;
569 
570                     //We are in the center cell
571                     if (x == p_row && y == p_col)
572                     {
573                         sub_matrix[sx, sy] = p_matrix[x, y];
574                         continue;
575                     }
576                     // ----- Corners ----- //
577 
578                     //Up-Left
579                     if (x < 0 && y < 0 && (p_matrix[max_x + x, max_y + y
```

```
                ] != DEAD))
580             {
581                 sub_matrix[sx, sy] = p_matrix[max_x + x, max_y +
                        y];
582             }
583             //Down-Right
584             if (x >= max_x && y >= max_y && (p_matrix[x - max_x,
                    y - max_y] != DEAD))
585             {
586                 sub_matrix[sx, sy] = p_matrix[x - max_x, y -
                        max_y];
587             }
588             //Up-Right
589             if (x < 0 && y == max_y && (p_matrix[max_x + x, 0]
                    != DEAD))
590             {
591                 sub_matrix[sx, sy] = p_matrix[max_x + x, 0];
592             }
593             //Down-left
594             if (x >= max_x && y < 0 && (p_matrix[x - max_x,
                    max_y + y] != DEAD))
595             {
596                 sub_matrix[sx, sy] = p_matrix[x - max_x, max_y +
                        y];
597             }
598             // ----- Edges ----- //
599
600             if (y >= 0 && y < max_y)
601             {
602                 //Up
603                 if (x < 0 && p_matrix[x + max_x, y] != DEAD)
604                 {
605                     sub_matrix[sx, sy] = p_matrix[x + max_x, y];
606                 }
607                 //Down
608                 else if (x >= max_x && p_matrix[x - max_x, y] !=
                        DEAD)
609                 {
610                     sub_matrix[sx, sy] = p_matrix[x - max_x, y];
611                 }
612             }
613             if (x >= 0 && x < max_x)
614             {
615                 //Right
616                 if (y < 0 && p_matrix[x, y + max_y] != DEAD)
617                 {
618                     sub_matrix[sx, sy] = p_matrix[x, y + max_y];
619                 }
620                 //Left
621                 else if (y >= max_y && p_matrix[x, y - max_y] !=
                        DEAD)
622                 {
623                     sub_matrix[sx, sy] = p_matrix[x, y - max_y];
```

```
624                         }
625                     }
626
627                     if (x < 0 || x >= max_x)
628                     {
629                         continue;
630                     }
631
632                     if (y < 0 || y >= max_y)
633                     {
634                         continue;
635                     }
636
637                     if (p_matrix[x, y] != DEAD)
638                     {
639                         sub_matrix[sx, sy] = p_matrix[x, y];
640                     }
641
642                 }
643             }
644             return sub_matrix;
645         }
646
647         private void storePatterns(int dimension) {
648             try {
649                 SaveFileDialog saveFileDialog = new SaveFileDialog();
650                 saveFileDialog.Filter = "Archivo de texto|*.txt";
651                 saveFileDialog.Title = "Patterns found " + dimension + "
                        x" + dimension;
652                 saveFileDialog.ShowDialog();
653                 StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
654                 sw.WriteLine("---------- Patterns found in a " +
                        dimension + " x " + dimension + " matrix ----------"
                        );
655                 foreach (KeyValuePair<ulong, ulong> item in patterns[
                        dimension]) {
656                     sw.WriteLine(item.Key + " appears " + item.Value + "
                            times ");
657                 }
658                 sw.Close();
659             }
660             catch (Exception e) {
661                 Console.WriteLine("An exception has occurred on
                        storePatterns() " + e);
662             }
663         }
664
665         /// <summary>
666         /// This function intialize our predefined patterns. This
                patterns
667         /// allow us to draw and drop patterns into the automata space.
668         /// </summary>
```

```
669         private void initMosaics()
670         {
671
672             figure.Add(null);
673
674             figure.Add(new uint[,] { { ALIVE, ALIVE},
675                                      { ALIVE, ALIVE } });
676
677             figure.Add(new uint[,] { { ALIVE, ALIVE, DEAD, ALIVE, ALIVE
                    },
678                                      { ALIVE, DEAD, DEAD, DEAD, ALIVE },
679                                      { DEAD, ALIVE, ALIVE, ALIVE, DEAD }
                                         });
680
681             figure.Add(new uint[,] { { ALIVE, ALIVE, DEAD },
682                                      { ALIVE, DEAD, ALIVE },
683                                      { DEAD, ALIVE, ALIVE } });
684
685             figure.Add(new uint[,] { { DEAD, ALIVE, DEAD },
686                                      { DEAD, DEAD, ALIVE },
687                                      { ALIVE, ALIVE, ALIVE } });
688
689             figure.Add(new uint[,] { { DEAD, ALIVE, DEAD },
690                                      { ALIVE, DEAD, ALIVE},
691                                      { DEAD, ALIVE, DEAD} });
692
693             figure.Add(new uint[,] { { DEAD, ALIVE, ALIVE, ALIVE, ALIVE,
                     ALIVE, ALIVE },
694                                      { ALIVE, DEAD, DEAD, DEAD, DEAD,
                                         DEAD, ALIVE },
695                                      { DEAD, DEAD, DEAD, DEAD, DEAD,
                                         DEAD, ALIVE},
696                                      { ALIVE, DEAD, DEAD, DEAD, DEAD,
                                         ALIVE, DEAD },
697                                      { DEAD, DEAD, ALIVE, ALIVE, DEAD,
                                         DEAD, DEAD } });
698
699             figure.Add(new uint[,] { { ALIVE, DEAD, DEAD },
700                                      { DEAD, DEAD, ALIVE },
701                                      { DEAD, DEAD, ALIVE },
702                                      { ALIVE, DEAD, DEAD } });
703
704             figure.Add(new uint[,] { { DEAD, ALIVE, ALIVE, DEAD },
705                                      { ALIVE, DEAD, DEAD, DEAD },
706                                      { ALIVE, DEAD, DEAD, DEAD },
707                                      { DEAD, ALIVE, ALIVE, DEAD } });
708
709             figure.Add(new uint[,] { { ALIVE, ALIVE, ALIVE } });
710
711             figure.Add(new uint[,] { { DEAD, DEAD, ALIVE },
712                                      { ALIVE, DEAD, DEAD },
713                                      { ALIVE, DEAD, DEAD },
714                                      { DEAD, ALIVE, DEAD } });
```

```
715
716         }
717
718         /// <summary>
719         /// Sort the elements of out Dictionary called data.
720         /// The elements are sorted using the Value
721         /// </summary>
722         private void sortTransitions()
723         {
724
725             foreach (var item in data.OrderByDescending(key => key.Value
                    ))
726             {
727                 if (!recurrences.ContainsKey(item.Value))
728                 {
729                     List<ulong> aux = new List<ulong>();
730                     aux.Add(item.Key);
731                     recurrences.Add(item.Value, aux);
732                 }
733                 else
734                     recurrences[item.Value].Add(item.Key);
735             }
736
737         }
738
739         private void CreateGraphObjects()
740         {
741             //Itering into each element of the Dictionary
742             foreach (KeyValuePair<ulong, List<ulong>> item in
                    recurrences)
743             {
744                 Dictionary<ulong, List<ulong>> aux = new Dictionary<
                        ulong, List<ulong>>();
745                 aux.Add(item.Key, item.Value);
746                 //Itering through each element of the list
747                 for (int i = 0; i < item.Value.Count; i++)
748                 {
749                     ulong element_list = item.Value[i];
750                     if (recurrences.ContainsKey(element_list))
751                     {
752                         if (!aux.ContainsKey(element_list))
753                             aux.Add(element_list, recurrences[
                                    element_list]);
754                     }
755
756                 }
757                 Graph graph_element = new Graph(aux);
758                 //List<ulong> key = graph_element.getKey();
759                 string key = graph_element.getKey();
760                 if (aux.Count > 0 && !clasifications.ContainsKey(key))
761                     clasifications.Add(key, graph_element);
762             }
763         }
```

```
764
765         /// <summary>
766         /// Comer if two list are equal
767         /// </summary>
768         /// <param name="first_list">First list</param>
769         /// <param name="second_list">Second list</param>
770         /// <returns>true if the lists are equals</returns>
771         private bool compareTwoList(List<ulong> first_list, List<ulong>
                second_list)
772         {
773             if (first_list.Count != second_list.Count)
774                 return false;
775
776             for (int i = 0; i < first_list.Count; i++)
777             {
778                 if (first_list[i] != second_list[i])
779                     return false;
780             }
781             return true;
782         }
783
784         /// <summary>
785         /// Just for testing
786         /// </summary>
787         private void printRecurrences()
788         {
789             foreach (KeyValuePair<ulong, List<ulong>> item in
                    recurrences)
790             {
791                 Console.WriteLine("Element " + item.Key);
792                 for (int i = 0; i < item.Value.Count; i++)
793                 {
794                     Console.Write(item.Value[i] + " ");
795                 }
796                 Console.WriteLine();
797             }
798         }
799
800
801         /*****************************************************
802          *                        Events                     *
803          *****************************************************/
804
805         private void BTNStep_Click(object sender, EventArgs e)
806         {
807             step();
808         }
809
810         private void PBAutomataSimulator_MouseDown(object sender,
                MouseEventArgs e)
811         {
812             int x = e.X / cellArea;
813             int y = e.Y / cellArea;
```

```
814             matrix[x, y] = (matrix[x, y] == ALIVE) ? DEAD : ALIVE;
815             PBAutomataSimulator.Invalidate();
816         }
817
818         private void BTNStart_Click(object sender, EventArgs e)
819         {
820             if (BTNStart.Text == "Start")
821             {
822                 TimerSimulation.Start();
823                 BTNStart.Text = "Stop";
824             }
825             else
826             {
827                 TimerSimulation.Stop();
828                 BTNStart.Text = "Start";
829             }
830         }
831
832         private void trackBar1_ValueChanged(object sender, EventArgs e)
833         {
834             TimerSimulation.Interval = TBSpeed.Value;
835         }
836
837         private void TimerSimulation_Tick(object sender, EventArgs e)
838         {
839             step();
840         }
841
842         private void BTNZoomP_Click(object sender, EventArgs e)
843         {
844             if (cellArea < 50)
845             {
846                 cellArea++;
847                 PBAutomataSimulator.Invalidate();
848                 scrollBox();
849             }
850         }
851
852         private void BTNZoomM_Click(object sender, EventArgs e)
853         {
854             if (cellArea > 1)
855             {
856                 cellArea--;
857                 PBAutomataSimulator.Invalidate();
858                 scrollBox();
859             }
860         }
861
862         private void button1_Click(object sender, EventArgs e)
863         {
864             Random r = new Random();
865             for (int x = 0; x < matrix.GetLength(0); x++)
866             {
```

```
867
868                 for (int y = 0; y < matrix.GetLength(1); y++)
869                 {
870
871                     float rand = r.Next(0, 100);
872                     if (rand < double.Parse(numericOnes.Text))
873                     {
874                         matrix[x, y] = ALIVE;
875                     }
876                     else matrix[x, y] = DEAD;
877                 }
878             }
879             PBAutomataSimulator.Invalidate();
880         }
881
882         private void BTNClear_Click(object sender, EventArgs e)
883         {
884             CHHistogram.Series["#Ones"].Points.Clear();
885             for (int x = 0; x < matrix.GetLength(0); x++)
886             {
887
888                 for (int y = 0; y < matrix.GetLength(1); y++)
889                 {
890                     matrix[x, y] = DEAD;
891                 }
892             }
893             PBAutomataSimulator.Invalidate();
894             acumOnes = generation = 0;
895             for (int i = 0; i < patterns.Count; i++) {
896                 patterns[i].Clear();
897             }
898         }
899
900         private void BTNCreateMatrix_Click(object sender, EventArgs e)
901         {
902             int rows = (numericRows.Value == 0) ? 100 : (int)numericRows
                    .Value;
903             int cols = (numericCols.Value == 0) ? 100 : (int)numericCols
                    .Value;
904             createMatrix(rows, cols);
905         }
906
907         private void button2_Click(object sender, EventArgs e)
908         {
909             int min_lines = 0;
910             int min_chara = 0;
911             String fileName = null;
912
913             try
914             {
915                 using (OpenFileDialog openFileDialog = new
                        OpenFileDialog())
916                 {
```

```
917                     openFileDialog.InitialDirectory = "c\\";
918                     openFileDialog.Filter = "txt files (*.txt)|*.txt";
919                     openFileDialog.FilterIndex = 2;
920                     if (openFileDialog.ShowDialog() == DialogResult.OK)
921                     {
922                         fileName = openFileDialog.FileName;
923                     }
924                 }
925 
926                 if (fileName != null)
927                 {
928                     Console.WriteLine(fileName);
929                     StreamReader objectReader = new StreamReader(
                            fileName);
930                     //Reading the file, line per line
931                     String line = "";
932                     ArrayList arrayText = new ArrayList();
933                     while (line != null)
934                     {
935                         line = objectReader.ReadLine();
936                         if (line != null)
937                             arrayText.Add(line);
938                     }
939                     objectReader.Close();
940                     //Iterate into the ArrayList and send the
                            information to the GUI
941                     min_lines = arrayText.Count;
942                     min_chara = arrayText[0].ToString().Length;
943 
944                     Console.WriteLine(min_chara);
945                     Console.WriteLine(min_lines);
946                     if (min_chara > matrix.GetLength(1) && min_lines >
                            matrix.GetLength(0))
947                     {
948                         createMatrix(min_chara, min_lines);
949                     }
950                     for (int i = 0; i < min_lines; i++)
951                     {
952                         string strlne = arrayText[i].ToString().Trim();
953                         int j = 0;
954 
955                         foreach (char c in strlne)
956                         {
957                             matrix[j++, i] = (c == '1')?ALIVE:DEAD;
958                         }
959                     }
960                     Console.ReadLine();
961                 }
962             }
963             catch (Exception ex)
964             {
965                 Console.WriteLine(ex);
966             }
```

```
967             PBAutomataSimulator.Invalidate();
968         }
969
970         private void BTNSave_Click(object sender, EventArgs e)
971         {
972
973             try
974             {
975                 SaveFileDialog saveFileDialog = new SaveFileDialog();
976                 saveFileDialog.Filter = "Archivo de texto|*.txt";
977                 saveFileDialog.Title = "Actual state cellular automata";
978                 saveFileDialog.ShowDialog();
979                 if (saveFileDialog != null)
980                 {
981                     StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
982                     for (int i = 0; i < matrix.GetLength(0); i++)
983                     {
984
985                         for (int j = 0; j < matrix.GetLength(1); j++)
986                             sw.Write(((matrix[j, i] == DEAD) ? "1" : "0"
                                ));
987                         sw.WriteLine();
988                     }
989                     sw.Close();
990                     MessageBox.Show("I've stored something");
991                 }
992             }
993             catch (Exception ex)
994             {
995                 Console.WriteLine(ex);
996             }
997
998         }
999
1000        private void PBAutomataSimulator_MouseMove(object sender,
            MouseEventArgs e)
1001        {
1002            try
1003            {
1004                if (move && index_pattern != 0)
1005                {
1006                    int x = e.X / cellArea;
1007                    int y = e.Y / cellArea;
1008
1009                    uint[,] draw_figure = figure[index_pattern];
1010                    int x_size = (draw_figure).GetLength(1);
1011                    int y_size = (draw_figure).GetLength(0);
1012                    initMosaics();
1013                    for (int x_c = 0; x_c < x_size; x_c++)
1014                    {
1015
1016                        for (int y_c = 0; y_c < y_size; y_c++)
```

```
1017                        {
1018                            matrix[x_c + x, y_c + y] = draw_figure[y_c,
                                    x_c];
1019                        }
1020                    }
1021                    PBAutomataSimulator.Invalidate();
1022                    move = false;
1023                    index_pattern = 0;
1024                }
1025            }
1026            catch (Exception) {
1027
1028            }
1029        }
1030
1031        private void pictureBox1_MouseUp(object sender, MouseEventArgs e
                )
1032        {
1033            move = true;
1034            index_pattern = 1;
1035        }
1036
1037        private void pictureBox2_MouseUp(object sender, MouseEventArgs e
                )
1038        {
1039            move = true;
1040            index_pattern = 2;
1041        }
1042
1043        private void pictureBox3_MouseUp(object sender, MouseEventArgs e
                )
1044        {
1045            move = true;
1046            index_pattern = 3;
1047        }
1048
1049        private void pictureBox4_MouseUp(object sender, MouseEventArgs e
                )
1050        {
1051            move = true;
1052            index_pattern = 4;
1053        }
1054
1055        private void pictureBox5_MouseUp(object sender, MouseEventArgs e
                )
1056        {
1057            move = true;
1058            index_pattern = 5;
1059        }
1060
1061        private void pictureBox6_MouseUp(object sender, MouseEventArgs e
                )
1062        {
```

```
1063            move = true;
1064            index_pattern = 6;
1065        }
1066
1067        private void pictureBox7_MouseUp(object sender, MouseEventArgs e
                )
1068        {
1069            move = true;
1070            index_pattern = 7;
1071        }
1072
1073        private void pictureBox8_MouseUp(object sender, MouseEventArgs e
                )
1074        {
1075            move = true;
1076            index_pattern = 8;
1077        }
1078
1079        private void pictureBox9_MouseUp(object sender, MouseEventArgs e
                )
1080        {
1081            move = true;
1082            index_pattern = 9;
1083        }
1084
1085        private void pictureBox10_MouseUp(object sender, MouseEventArgs
                e)
1086        {
1087            move = true;
1088            index_pattern = 10;
1089        }
1090
1091        private void button2_Click_1(object sender, EventArgs e)
1092        {
1093            Color color = getColor();
1094            alive = new SolidBrush(color);
1095            ALIVE = ColorHandler.fromColorToInt(color);
1096            initMosaics();
1097            PBAutomataSimulator.Invalidate();
1098        }
1099
1100        private void BTNDeadCells_Click(object sender, EventArgs e)
1101        {
1102            dead = new SolidBrush(getColor());
1103            initMosaics();
1104            PBAutomataSimulator.Invalidate();
1105        }
1106
1107        private void BTNGrid_Click(object sender, EventArgs e)
1108        {
1109            grid = new Pen(getColor());
1110            PBAutomataSimulator.Invalidate();
1111        }
```

```
1112
1113        private void CBPoints_CheckedChanged(object sender, EventArgs e)
1114        {
1115            if (CBPoints.Checked)
1116                CHHistogram.Series[0].ChartType = System.Windows.Forms.
                    DataVisualization.Charting.SeriesChartType.Point;
1117            else
1118                CHHistogram.Series[0].ChartType = System.Windows.Forms.
                    DataVisualization.Charting.SeriesChartType.Column;
1119        }
1120
1121        private void generatePatternsToolStripMenuItem_Click(object
            sender, EventArgs e)
1122        {
1123            try
1124            {
1125                string value = Microsoft.VisualBasic.Interaction.
                    InputBox("Write the dimension of the matrix", "
                    Generate patterns", "", 0, 0);
1126                //Generating patterns
1127                if (Convert.ToInt16(value) <= 4)
1128                    generatePatterns(Convert.ToInt16(value));
1129                else
1130                    MessageBox.Show("I'm sorry but the maximum size it's
                        4 :c");
1131            }
1132            catch (Exception ex) {
1133                Console.WriteLine("An exception has occured capturing
                    text to generate patterns " + ex);
1134            }
1135        }
1136
1137        private void BTNSavePatterns_Click(object sender, EventArgs e)
1138        {
1139            if (CBPatternRecognition.Checked)
1140            {
1141                for (int i = 2; i <= 4; i++)
1142                {
1143                    storePatterns(i);
1144                }
1145            }
1146            else
1147                MessageBox.Show("No se han guardado datos hasta ahora");
1148        }
1149    }
1150 }
```

1.8 Conclusiones

Esta práctica es bastante interesante, ya que es increíble lo que se puede lograr con un par de condiciones, que a simple vista parecen insignificantes. Es posible el

observar como es que una sola célula puede causar un gran caos en todo el sistema, claro está eso dependerá de las condiciones que sean asignadas. Esto lo podemos ver con las reglas de difusión, es posible crear figuras increíbles con tan solo un par de elementos, como los gliders, los cuales al colisionar generan una figura que prácticamente es infinita y de la cual pude observar que claramente era un fractal. También es importante recalcar que tanto en "Game of life" y "Diffusion" encontramos algunas figuras que se vuelven periódicas, como los osciladores, los ya mencionados gliders, y algunas otras figuras.