Computing selected topics

- Prácticas tercer 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

Un autómata celular es un modelo matemático, el cual describe a un sistema dinámico el cual evoluciona en pasos discretos. Estos autómatas celulares son perfectos para modelar sistemas naturales los cuales pueden ser descritos como una colección masiva de objetos simples que interactúen localmente unos con otros.

1.2 Historia

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.

Figure 1.1: John von Neumann

1.3 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 conjunto 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.

1.4 Sistemas dinámicos

Un sistema dinámico es un sistema cuyo estado evoluciona a través del tiempo. Los sistemas físicos en estado no estacionario son ejemplos de sistemas dinámicos, pero también es aplicable a modelos económicos, matemáticos y de otros tipos.

1.5 Componentes de un autómata celular

1.5.1 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.

1.5.2 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.

1.5.3 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.

1.5.4 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.

1.6 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.

1.6.1 Frontera abierta

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

1.6.2 Frontera reflectora

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

1.6.3 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.

1.6.4 Sin frontera

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

1.7 Tipos de autómatas celulares

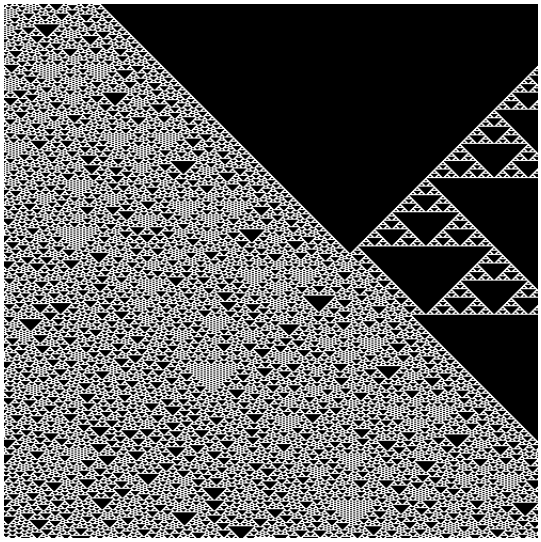
Cuando hablamos de autómatas celulares, nos referimos a pequeñas entidades independientes pero que interaccionan entre si. Son llamados celulares porque son la unidad elemental del universo donde van a existir y autómatas porque deciden por si mismos, basados en un conjunto de reglas bien definidas. Existe un universo de autómatas celulares multidimensionales, pero podemos extraer pequeños fragmentos de ese gran universo.

1.7.1 Autómata celular 1-D

O bien autómatas celulares unidimensionales, son una secuencia lineal de n células, $\{a_1, a_2, a_3, ..., a_n\}$ donde a_1 y a_n son adyacentes entre si. El vecindario de radio r de una célula c está formado por el conjunto de aquellas células que se encuentran a distancia r de c.

Un ejemplo En la siguiente imagen podemos observar cierto patrón generado con un autómata celular unidimensional

Figure 1.2: Autómata celular unidimensional regla 126

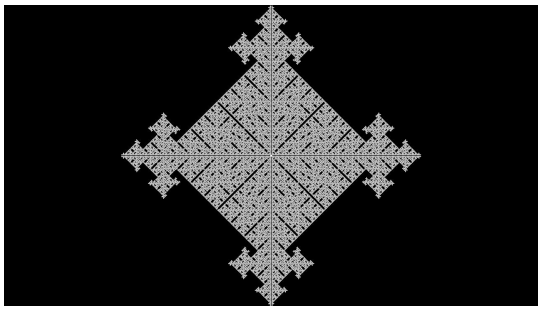


1.7.2 Autómata celular 2-D

Como sabemos los autómatas celulares pueden ser unidimensionales, como ya se ha descrito, pero también pueden ser en dos dimensiones. Para estos autómatas cada célula tiene ocho vecinos si consideramos las células que estan en las diagonales, o bien cuatro vecinos si solo consideramos las células que están arriba, abajo, a la izquierda y a la derecha.

Un ejemplo En la siguiente imagen podemos observar cierto patrón generado con un autómata celular bidimensional

Figure 1.3: Autómata celular bidimensional



1.8 Práctica

1.8.1 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.

Reconocimiento de patrones Además, 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.

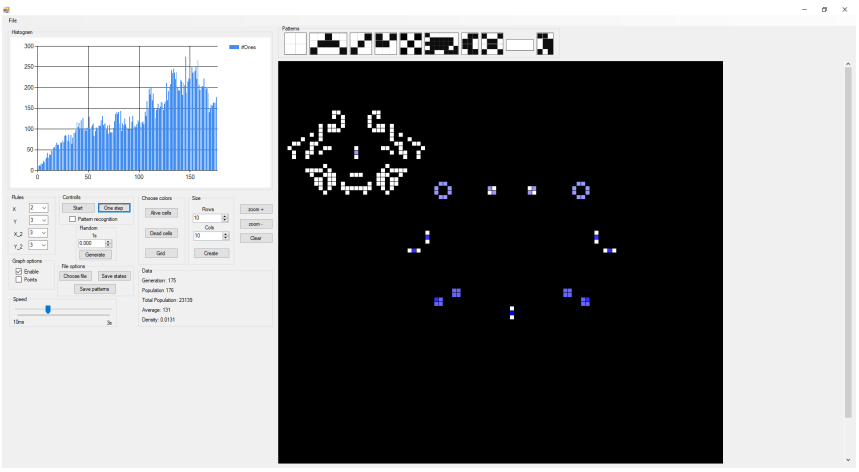
Autómata celular con memoria Como sabemos los autómatas celulares no tiene memoria, los nuevos estados de las células dependen de la configuración de la vecindad, este procedimiento se realiza en un paso. Un autómata celular con memoria es una extensión de un autómata celular. Cada célula c_i puede recordar sus estados durante cierto periodo fijo de evolución.

Dicho lo anterior, también se aplicado esta técnica para obtener información adicional sobre el autómata celular. En este caso se ha aplicado la técnica en el juego de la vida de John Conway.

1.8.2 Pruebas

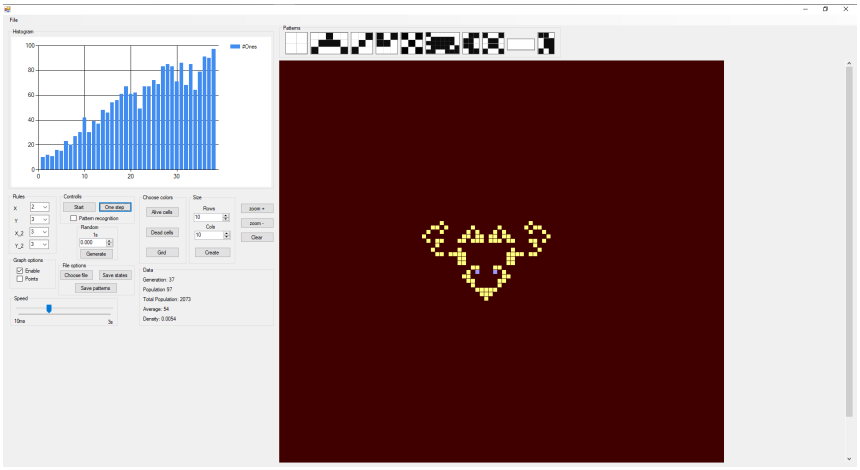
Este programa tiene varios aspectos que han sido cubiertos. Para iniciar las pruebas se han añadido algunas células vivas haciendo click en el elemento que inicialmente es color negro. Esto se realiza haciendo uso de algunos eventos, tal y como se muestra en la siguiente figura.

Figure 1.4: Prueba haciendo uso de eventos



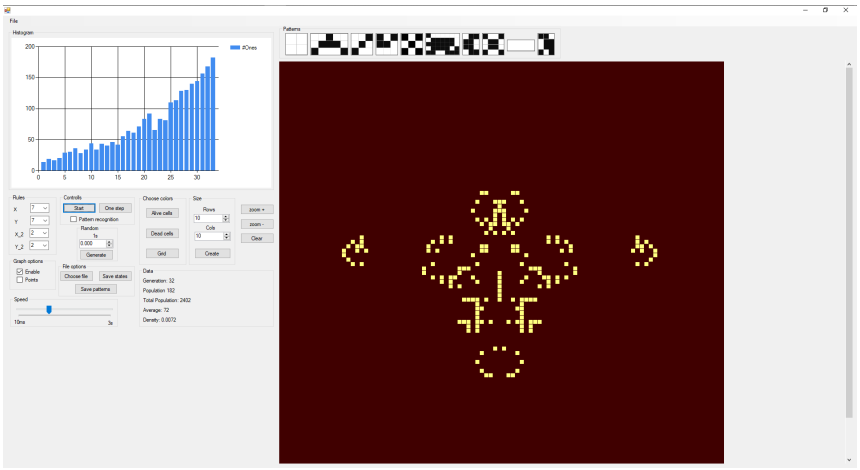
A continuación se muestra el uso del cambio de colores. Podemos seleccionar cualquiera de los colores RGB disponibles, estos podemos aplicarlos tanto al color de células muertas, como células vivas y el grid que tiene la matriz

Figure 1.5: 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 1.6: 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 1.7: Guardando datos

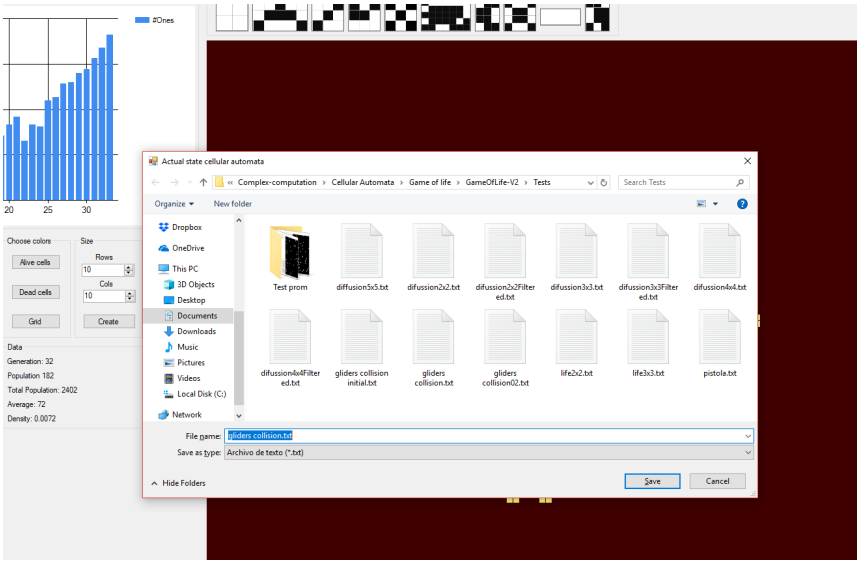
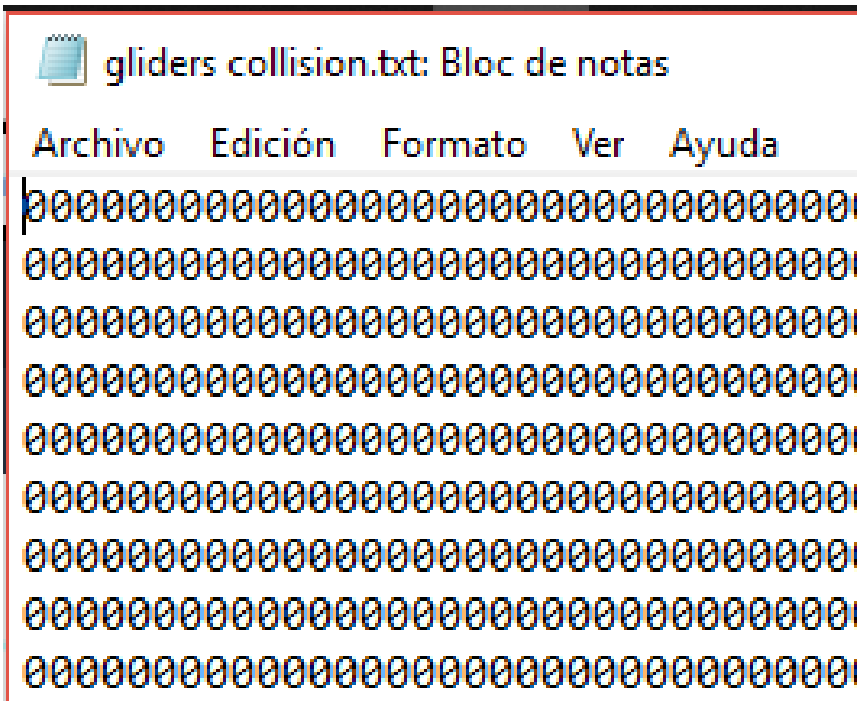
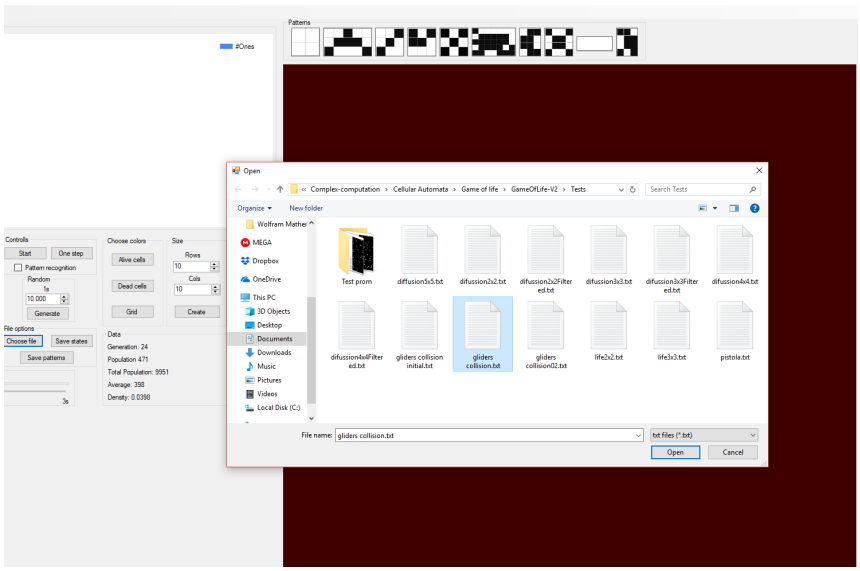


Figure 1.8: Contenido del archivo



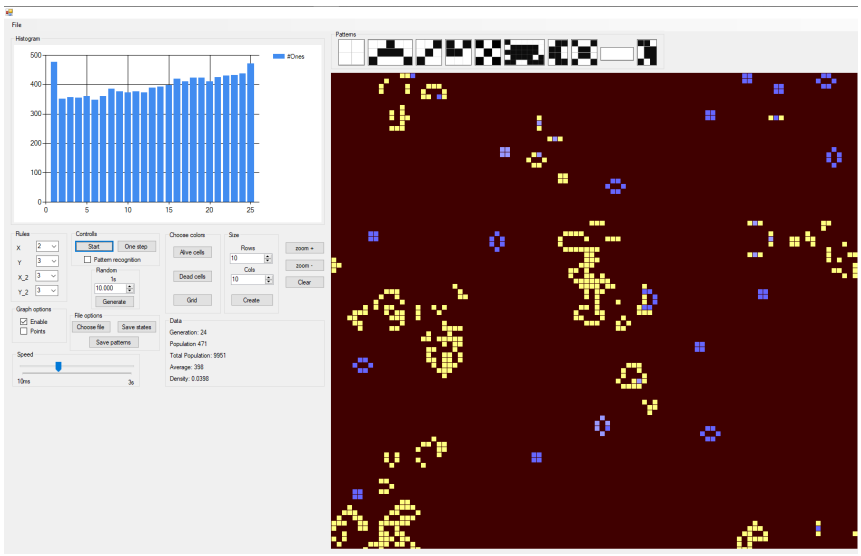
También como era de esperarse algún archivo de texto también puede ser cargado al programa.

Figure 1.9: 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 1.10: Random



1.9 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 1.11: Grafos según la regla "Game of life" archivo original matriz 3x3

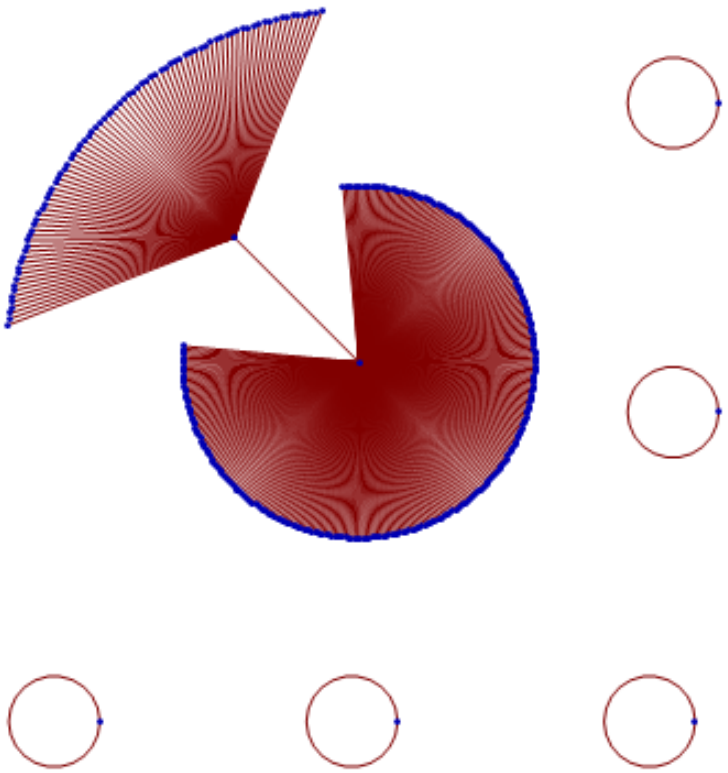
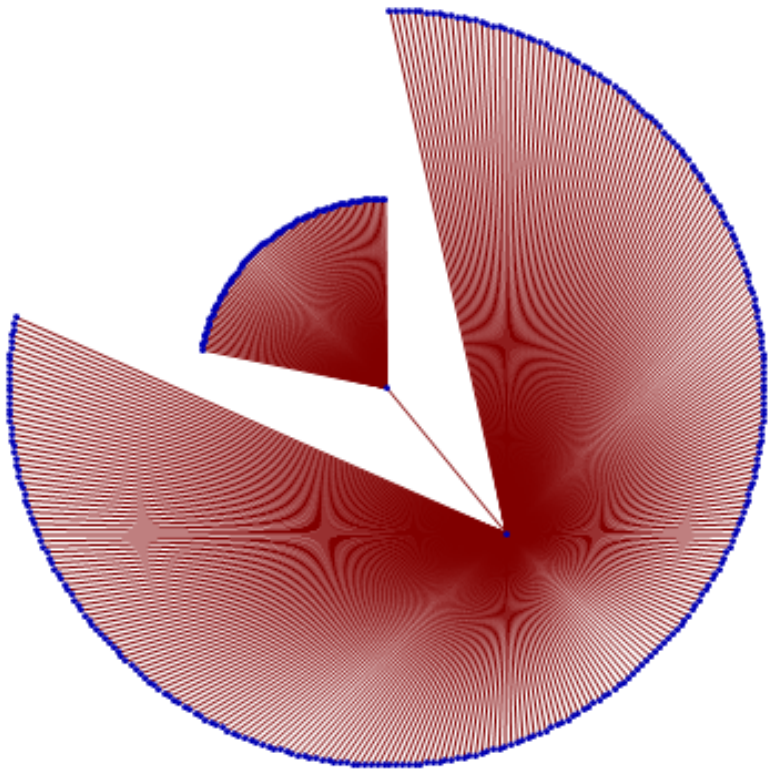
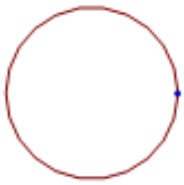


Figure 1.12: Grafos según la regla "Game of life" archivo filtrado matriz 3x3





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

Figure 1.13: Grafos según la regla "Game of life" archivo original matriz 2x2

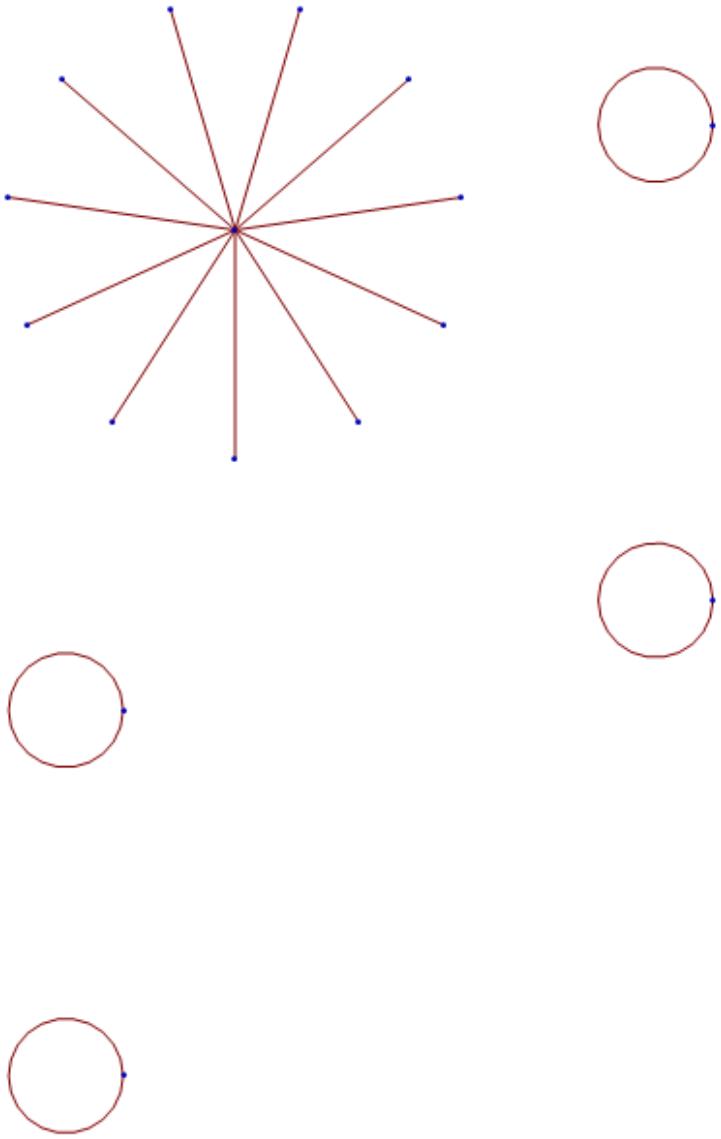
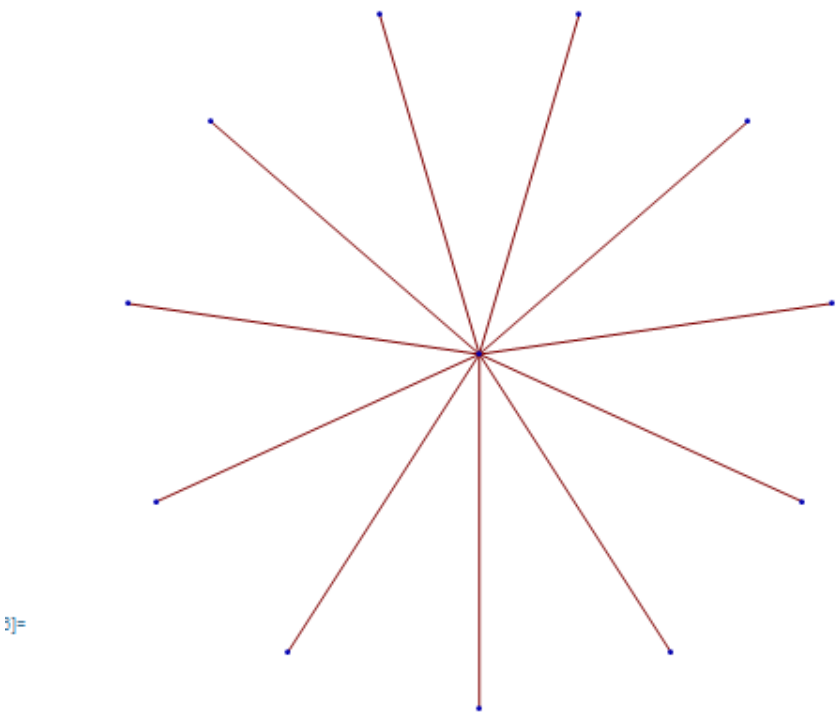


Figure 1.14: Grafos según la regla "Game of life" archivo filtrado matriz 2x2



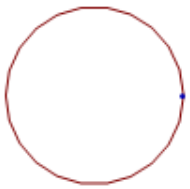


Figure 1.15: 01. Grafos según la regla "Game of life" archivo original matriz 4x4

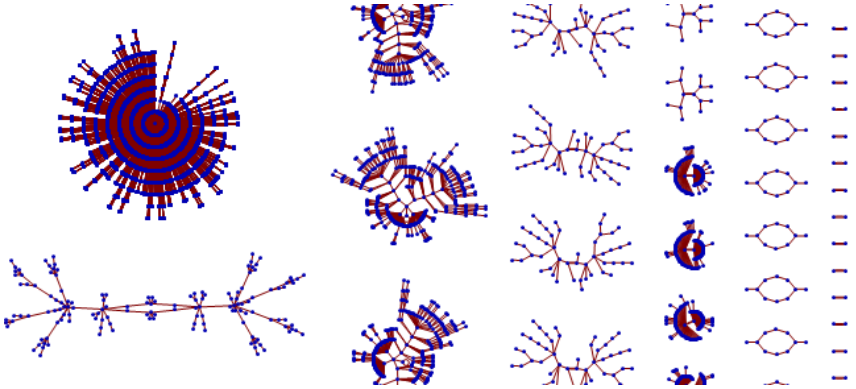


Figure 1.16: 02. Grafos según la regla "Game of life" archivo original matriz 4x4

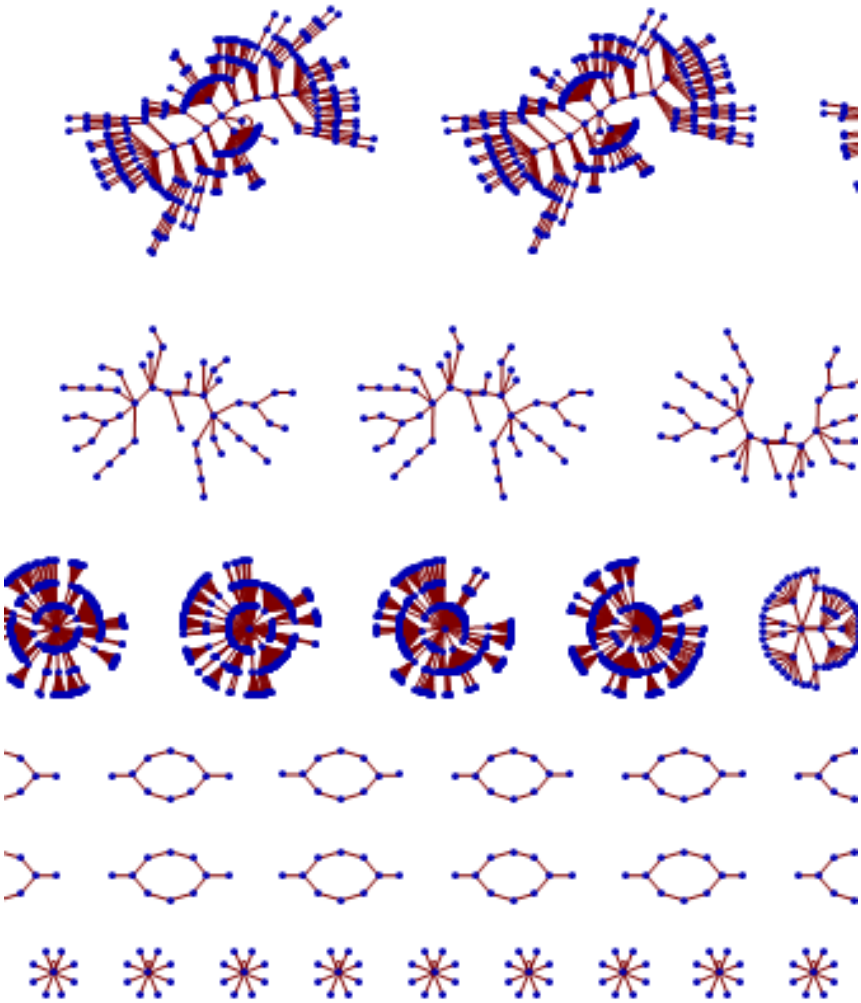
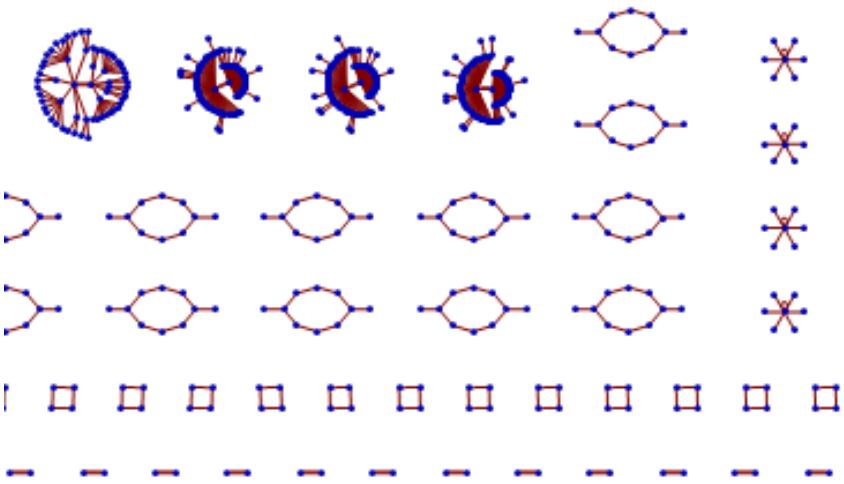
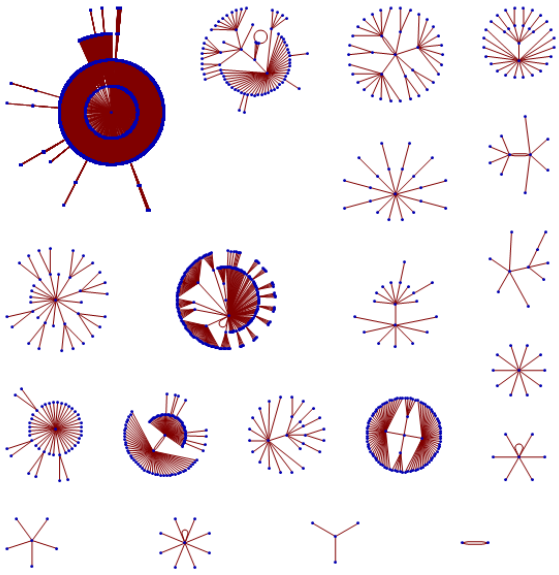


Figure 1.17: 03. Grafos según la regla "Game of life" archivo original matriz 4x4



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 1.18: 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 1.19: Grafos según la regla "Diffusion" archivo original matriz 2x2

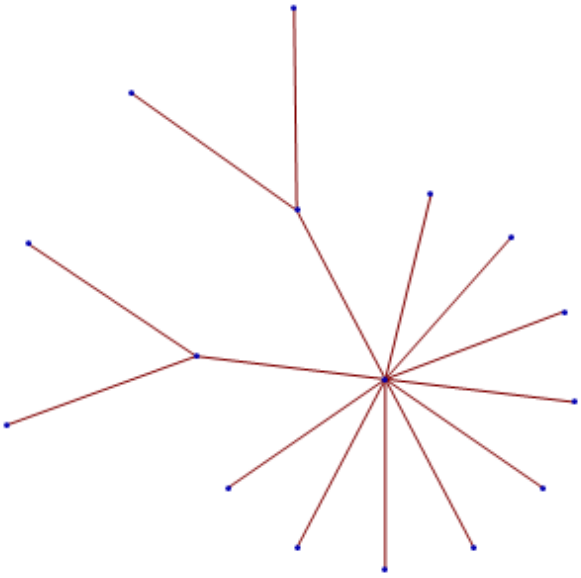


Figure 1.20: Grafos según la regla "Diffusion" archivo filtrado matriz 2x2

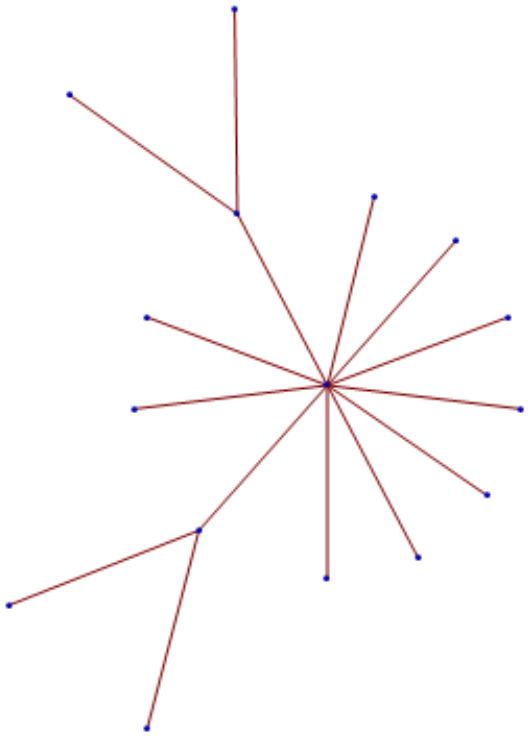


Figure 1.21: Grafos según la regla "Diffusion" archivo original matriz 3x3

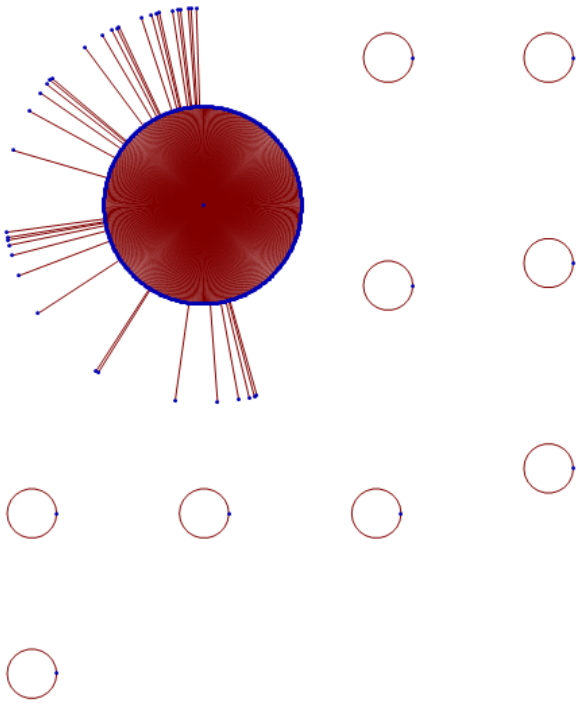


Figure 1.22: Grafos según la regla "Diffusion" archivo filtrado matriz 3x3

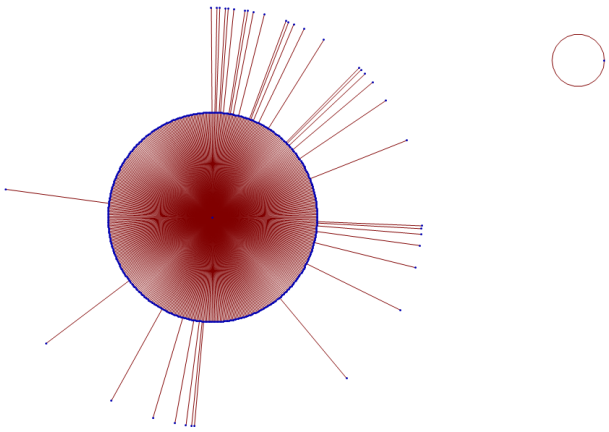


Figure 1.23: 01.Grafos según la regla "Diffusion" archivo original matriz 4x4

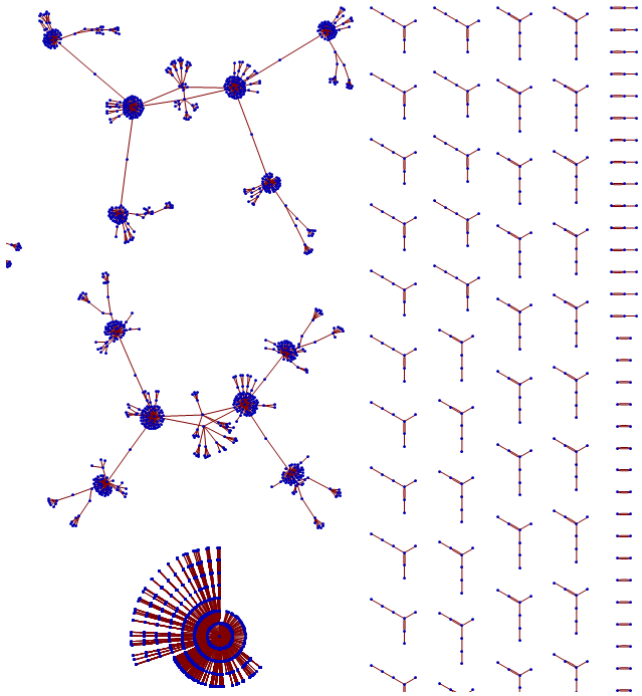


Figure 1.24: 02.Grafos según la regla "Diffusion" archivo original matriz 4x4

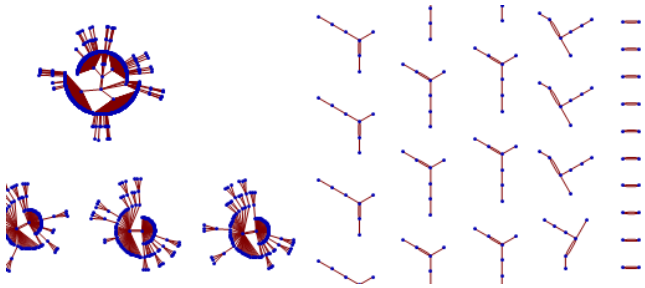
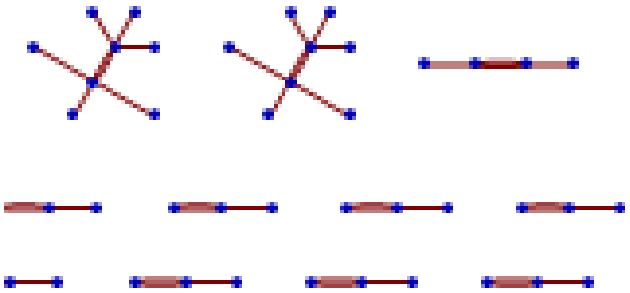
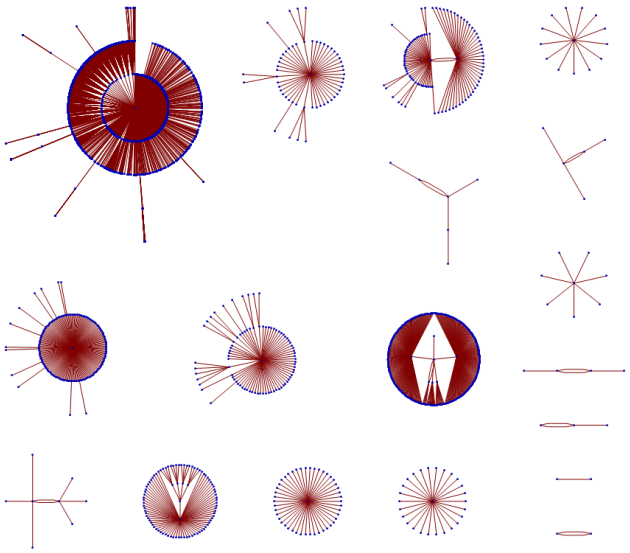


Figure 1.25: 03.Grafos según la regla "Diffusion" archivo original matriz 4x4



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

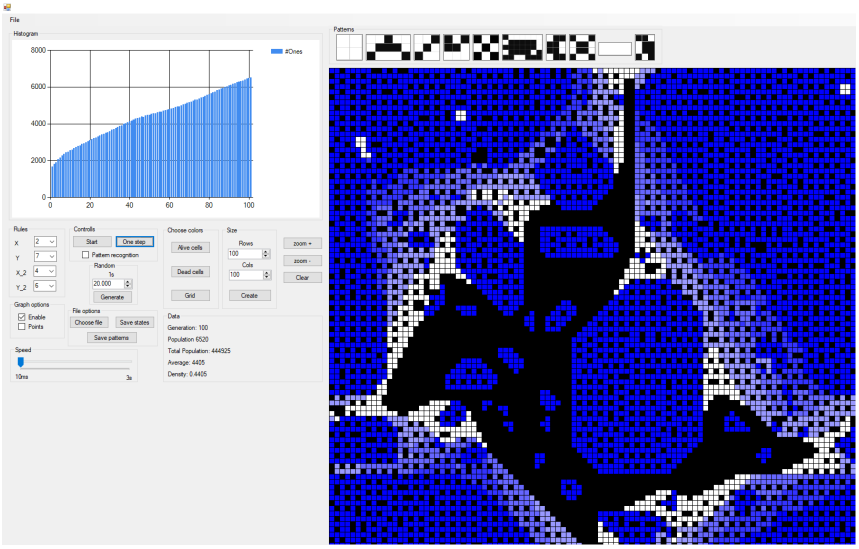
Figure 1.26: Grafos según la regla "Diffusion" archivo filtrado matriz 4x4



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"

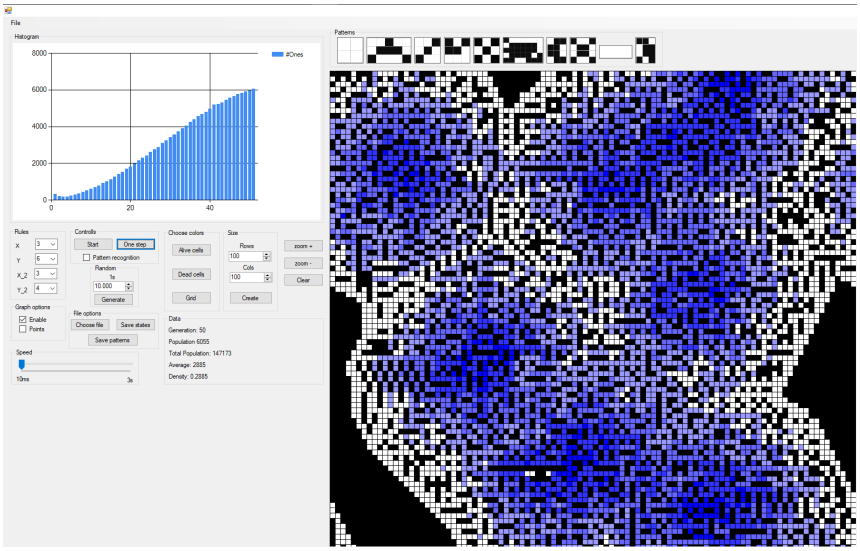
También se han probado más reglas en este autómata, primeramente se probó con la regla 2, 7, 4, 6 y una distribución de 20% de probabilidad en un espacio de cien por cien células. Además podemos apreciar el resultado al llegar a las 100 generaciones

Figure 1.27: Ejecución según la regla 2, 7, 4, 6



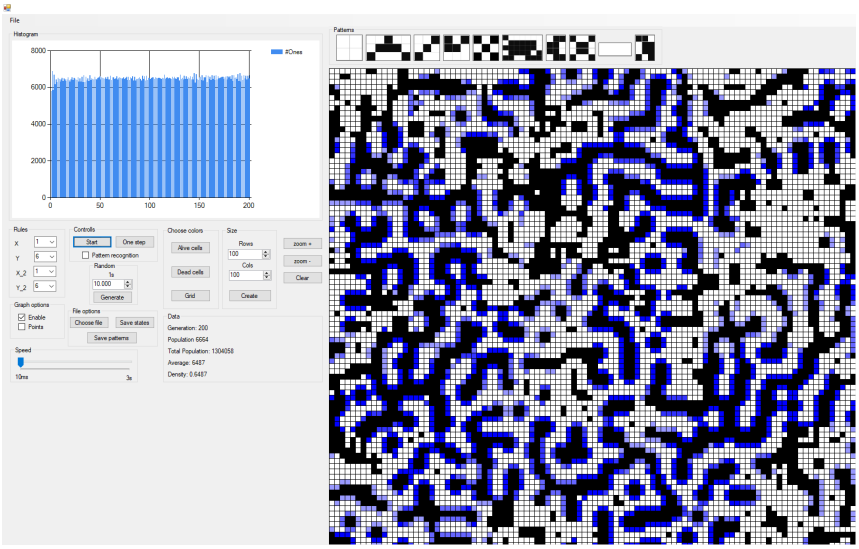
Otra regla que fue probada es la 3, 6, 3, 4 podemos ver que el comportamiento es similar a la anterior, y podemos apreciar esto en la gráfica, solo que para este caso se ejecutó el programa hasta la generación cincuenta y con una densidad en la distribución de células vivas menor.

Figure 1.28: Ejecución según la regla 3, 6, 3, 4



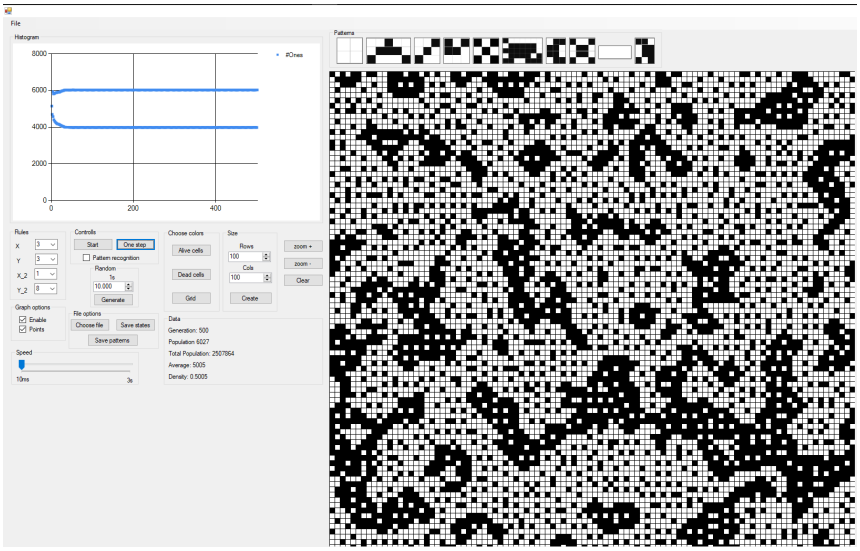
La siguiente regla que fue probada es la 1, 6, 1, 6 esta tiene un crecimiento bastante curioso, fue probada hasta la generación 200, y gracias a la función que se añadió del cambio de color cuando las células se han "estancado" podemos ver claramente que el comportamiento con esta regla no es de un oscilador.

Figure 1.29: 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"

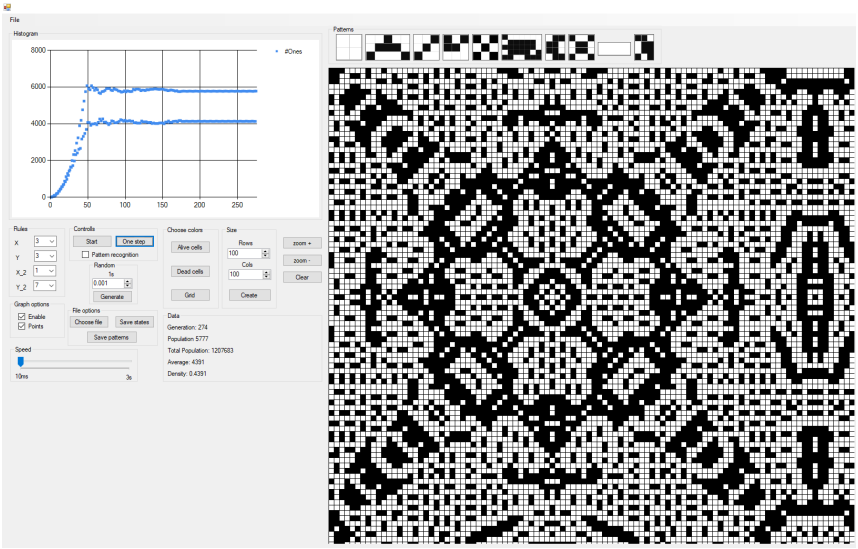
Figure 1.30: Ejecución según la regla 3, 3, 1, 8



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

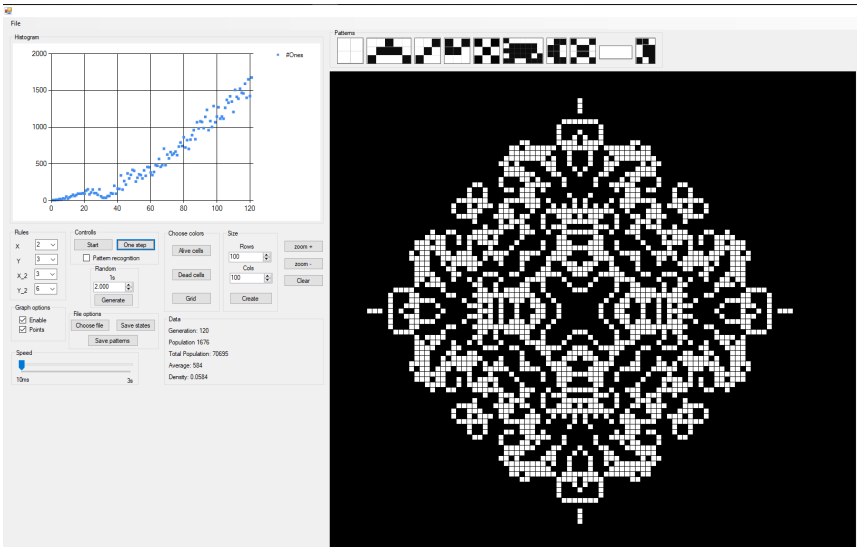
está lleno. La configuración inicial para esta regla fue un punto cerca del centro del espacio.

Figure 1.31: 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

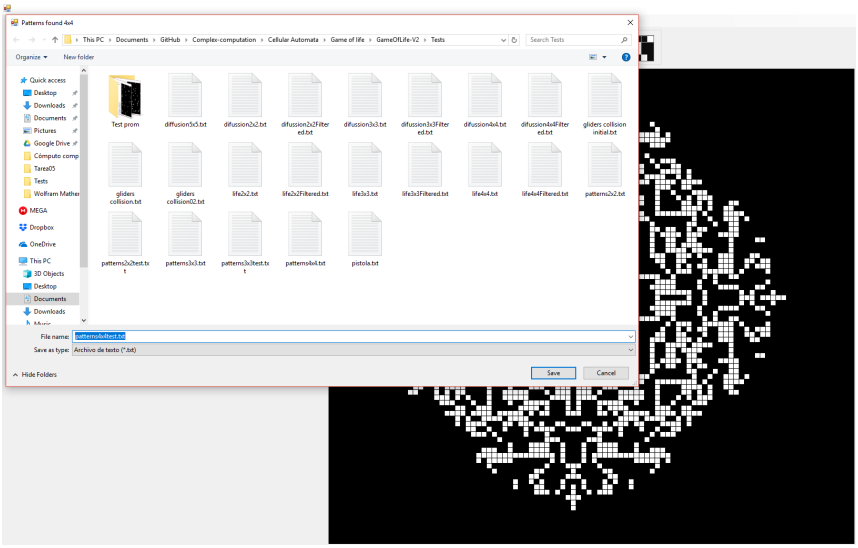
Figure 1.32: Ejecución según la regla 2, 3, 3, 6



1.10 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 1.33: Almacenamiento de los patrones

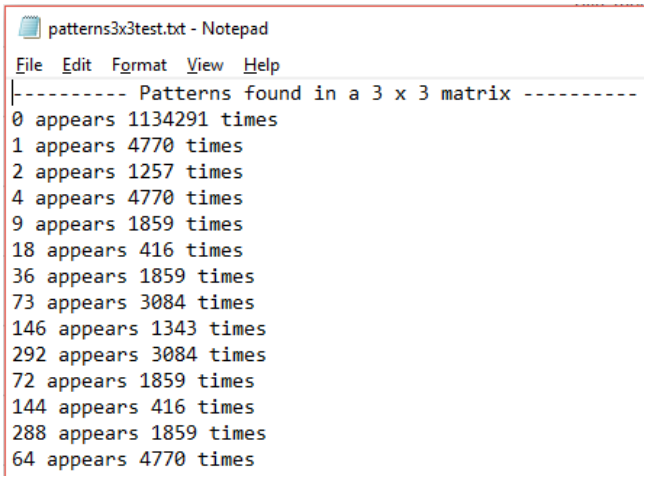


Una ves que se ha presionado el botón con la leyenda "Save patterns" se guardarán tres archivos, uno por cada tamaño de las submatrices evaluadas, en la siguiente imagen podemos apreciar los tres archivos generados

Figure 1.34: Archivos generados

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

Figure 1.35: fragmento del contenido del archivo patterns3x3test.txt



1.11 Autómata celular con memoria

Como ya se ha mencionado un autómata celular con memoria es una extensión de un autómata celular. Para estos autómatas celulares se debe almacenar información sobre los estados anteriores en la célula c_i en nuestro caso una simple matriz basto para poder almacenar esta información. El hacer uso de memoria en un autómata celular también nos brinda la posibilidad de utilizar más reglas, pero para nuestro caso solo usaremos tres:

- Mayoría: se coloca un uno en la celda si hay una cantidad estrictamente mayor de unos que de ceros obtenidos hasta la τ-ésima generación en caso contrario se coloca un cero
- Minoria: se coloca un uno en la celda si hay una cantidad estrictamente menor de unos que de ceros obtenidos hasta la τ-ésima generación en caso contrario se coloca un cero
- Paridad: se coloca un uno en la celda si y solo si hay la misma cantidad de unos que de ceros hasta la τ-ésima generación

Como podemos observar se debe definir una cantidad fija de generaciones de las cuales almacenaremos información para poder aplicar las reglas que se han descrito anteriormente. Podemos ver algunos ejemplos a continuación.

Figure 1.36: Configuración inicial

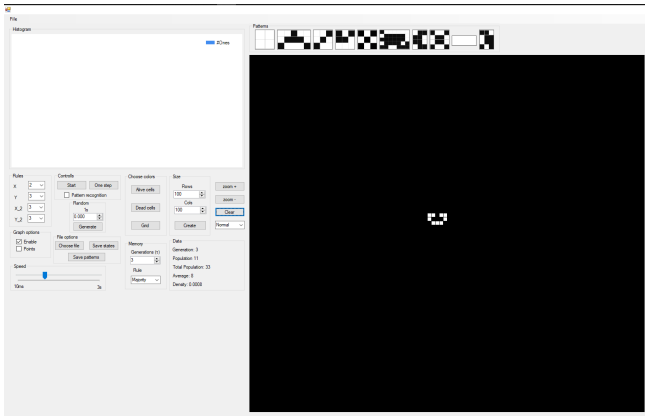


Figure 1.37: Aplicando regla de "life" hasta la tercer generación

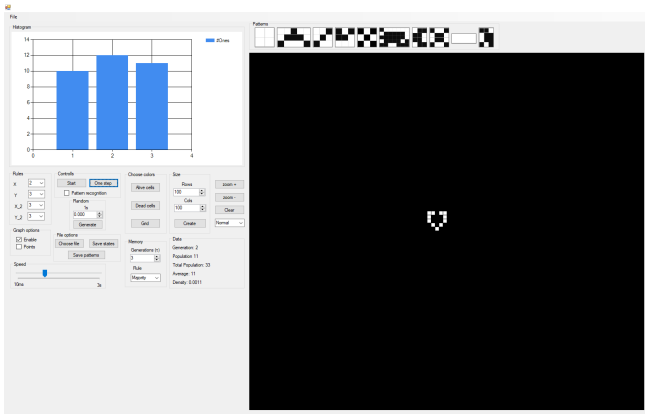


Figure 1.38: Uso de memoria hasta la tercer generación regla de mayoria

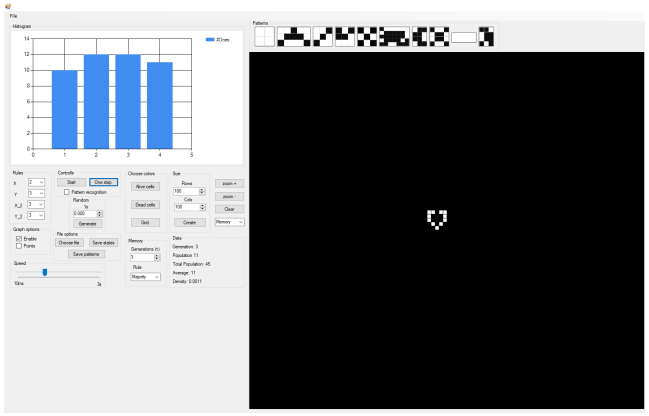


Figure 1.39: Uso de memoria hasta la tercer generación regla de minoria

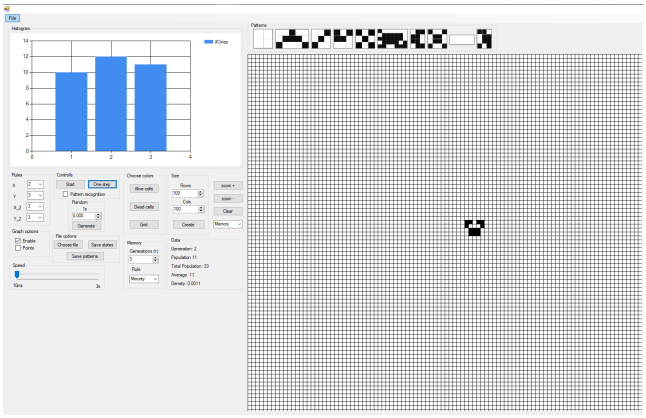


Figure 1.40: Aplicando regla de "life" hasta la sexta generación

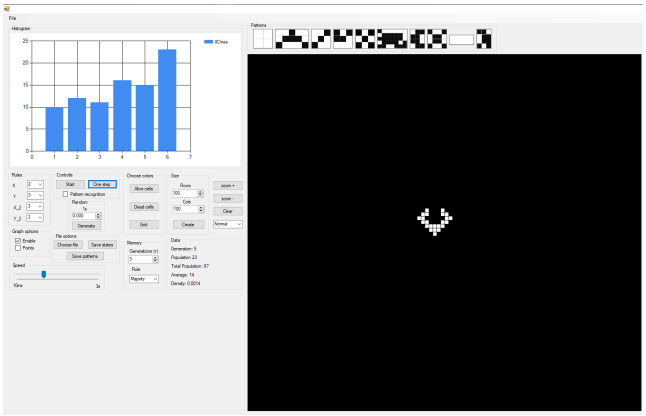


Figure 1.41: Uso de memoria hasta la sexta generación regla de mayoria

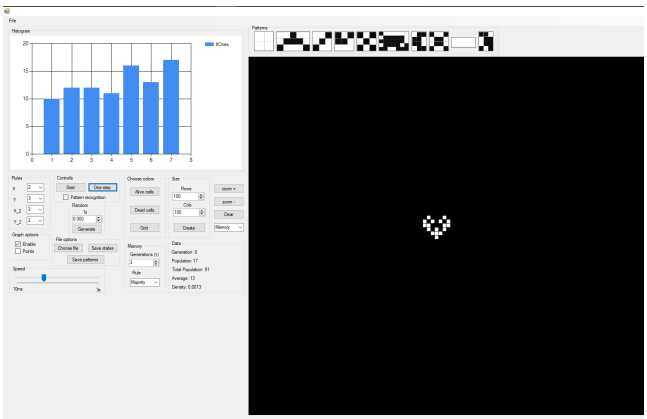
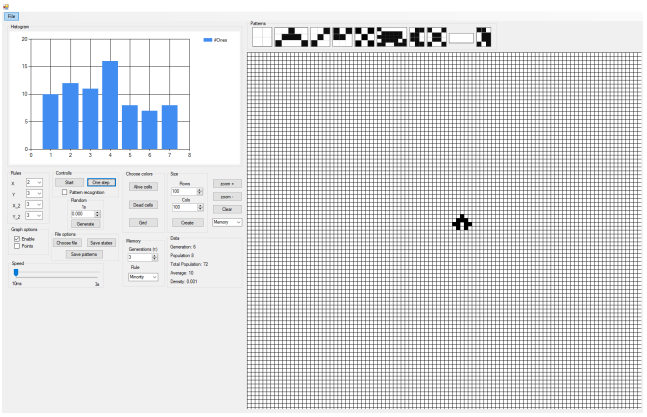


Figure 1.42: Uso de memoria hasta la sexta generación regla de minoria



Nos hace falta realizar una comparación similar pero utilizando un valor para τ par, esto con el fin de poder utilizar la regla de paridad de los autómatas celulares con memoria. Utilizaremos el mismo patrón inicial que el seleccionado anteriormente.

Figure 1.43: Uso de memoria hasta la cuarta generación regla de minoria

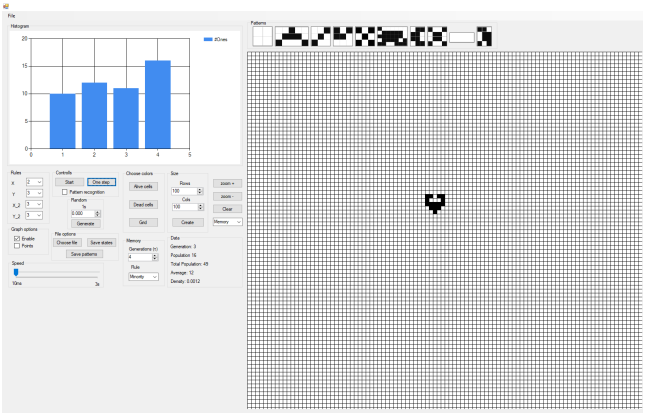


Figure 1.44: Uso de memoria hasta la cuarta generación regla de mayoria

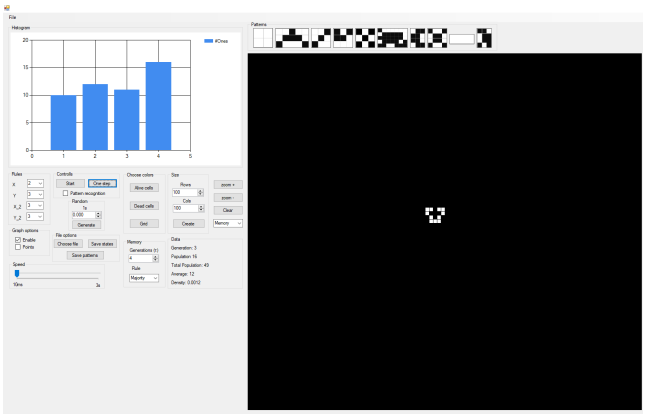


Figure 1.45: Uso de memoria hasta la cuarta generación regla de paridad

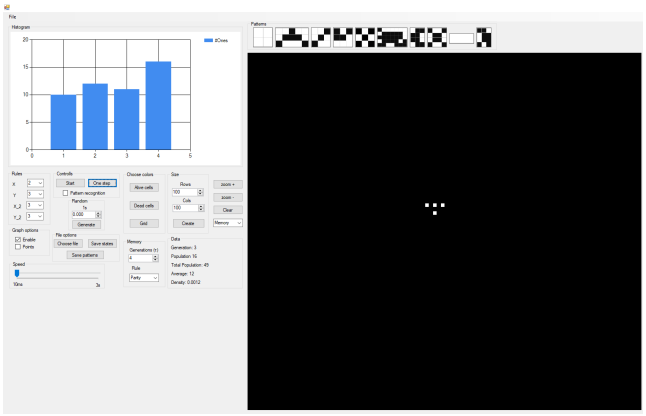


Figure 1.46: Uso de memoria hasta la octava generación regla de minoria

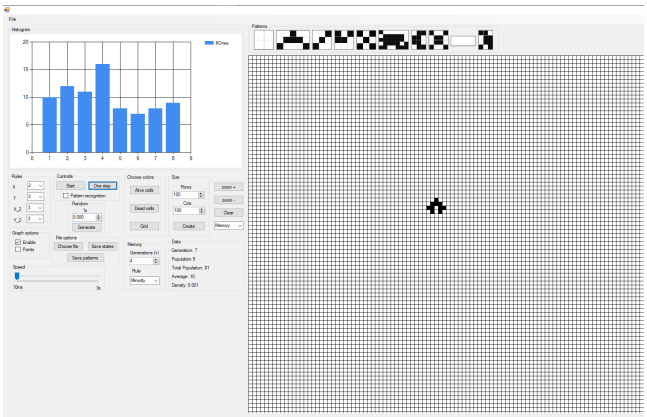


Figure 1.47: Uso de memoria hasta la octava generación regla de mayoria

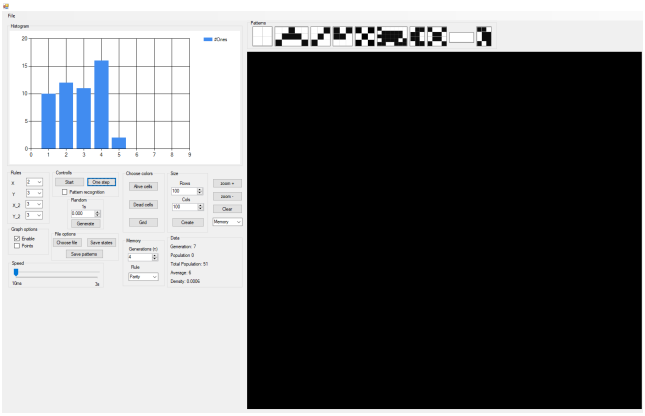
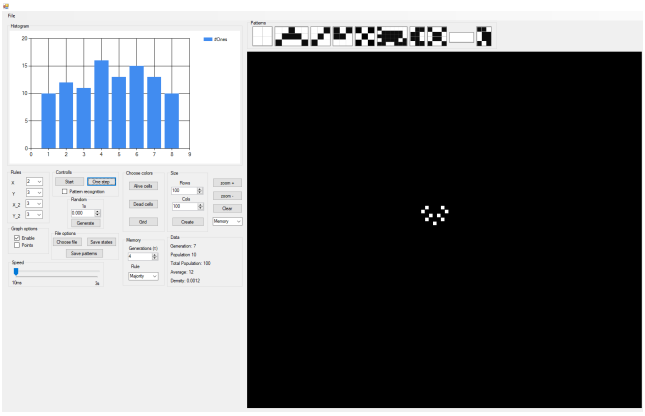


Figure 1.48: Uso de memoria hasta la octava generación regla de paridad



1.11.1 Código

A continuación se anexa 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
                   chartArea3 = new System.Windows.Forms.DataVisualization.
                   Charting.ChartArea();
33             System.Windows.Forms.DataVisualization.Charting.Legend
                   legend3 = new System.Windows.Forms.DataVisualization.
                   Charting.Legend();
34             System.Windows.Forms.DataVisualization.Charting.Series
                   series3 = 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.CBPatternRecognition = new System.Windows.Forms.
                    CheckBox();
62              this.groupBox5 = new System.Windows.Forms.GroupBox();
63              this.button1 = new System.Windows.Forms.Button();
64              this.numericOnes = new System.Windows.Forms.NumericUpDown();
65              this.label7 = new System.Windows.Forms.Label();
66              this.groupBox6 = new System.Windows.Forms.GroupBox();
67              this.BTNGrid = new System.Windows.Forms.Button();
68              this.BTNDeadCells = new System.Windows.Forms.Button();
69              this.BTNAliveCells = new System.Windows.Forms.Button();
70              this.BTNSelectFile = new System.Windows.Forms.Button();
71              this.BTNClear = new System.Windows.Forms.Button();
72              this.groupBox7 = new System.Windows.Forms.GroupBox();
73              this.BTNCreateMatrix = new System.Windows.Forms.Button();
74              this.numericCols = new System.Windows.Forms.NumericUpDown();
75              this.numericRows = new System.Windows.Forms.NumericUpDown();
76              this.label11 = new System.Windows.Forms.Label();
77              this.label8 = new System.Windows.Forms.Label();
78              this.groupBox8 = new System.Windows.Forms.GroupBox();
79              this.BTNSavePatterns = new System.Windows.Forms.Button();
80              this.BTNSave = new System.Windows.Forms.Button();
81              this.label12 = new System.Windows.Forms.Label();
82              this.label13 = new System.Windows.Forms.Label();
83              this.label14 = new System.Windows.Forms.Label();
84              this.groupBox9 = new System.Windows.Forms.GroupBox();
85              this.CBPoints = new System.Windows.Forms.CheckBox();
86              this.CheckGraphEnabled = new System.Windows.Forms.CheckBox()
                    ;
87              this.groupBox10 = new System.Windows.Forms.GroupBox();
88              this.pictureBox10 = new System.Windows.Forms.PictureBox();
89              this.pictureBox9 = new System.Windows.Forms.PictureBox();
```

```
90              this.pictureBox8 = new System.Windows.Forms.PictureBox();
91              this.pictureBox7 = new System.Windows.Forms.PictureBox();
92              this.pictureBox6 = new System.Windows.Forms.PictureBox();
93              this.pictureBox5 = new System.Windows.Forms.PictureBox();
94              this.pictureBox4 = new System.Windows.Forms.PictureBox();
95              this.pictureBox3 = new System.Windows.Forms.PictureBox();
96              this.pictureBox2 = new System.Windows.Forms.PictureBox();
97              this.pictureBox1 = new System.Windows.Forms.PictureBox();
98              this.groupBox11 = new System.Windows.Forms.GroupBox();
99              this.colorDialog = new System.Windows.Forms.ColorDialog();
100             this.menuStrip1 = new System.Windows.Forms.MenuStrip();
101             this.generarPatronesToolStripMenuItem = new System.Windows.
                    Forms.ToolStripMenuItem();
102             this.generatePatternsToolStripMenuItem = new System.Windows.
                    Forms.ToolStripMenuItem();
103             this.groupBox12 = new System.Windows.Forms.GroupBox();
104             this.label10 = new System.Windows.Forms.Label();
105             this.label9 = new System.Windows.Forms.Label();
106             this.comboRuleMem = new System.Windows.Forms.ComboBox();
107             this.NumericGenMem = new System.Windows.Forms.NumericUpDown
                    ();
108             this.backgroundWorker1 = new System.ComponentModel.
                    BackgroundWorker();
109             this.comboSpace = new System.Windows.Forms.ComboBox();
110             ((System.ComponentModel.ISupportInitialize)(this.
                    PBAutomataSimulator)).BeginInit();
111             ((System.ComponentModel.ISupportInitialize)(this.CHHistogram
                    )).BeginInit();
112             this.groupBox1.SuspendLayout();
113             this.groupBox2.SuspendLayout();
114             ((System.ComponentModel.ISupportInitialize)(this.TBSpeed)).
                    BeginInit();
115             this.flowLayoutPanel1.SuspendLayout();
116             this.groupBox3.SuspendLayout();
117             this.groupBox4.SuspendLayout();
118             this.groupBox5.SuspendLayout();
119             ((System.ComponentModel.ISupportInitialize)(this.numericOnes
                    )).BeginInit();
120             this.groupBox6.SuspendLayout();
121             this.groupBox7.SuspendLayout();
122             ((System.ComponentModel.ISupportInitialize)(this.numericCols
                    )).BeginInit();
123             ((System.ComponentModel.ISupportInitialize)(this.numericRows
                    )).BeginInit();
124             this.groupBox8.SuspendLayout();
125             this.groupBox9.SuspendLayout();
126             this.groupBox10.SuspendLayout();
127             ((System.ComponentModel.ISupportInitialize)(this.
                    pictureBox10)).BeginInit();
128             ((System.ComponentModel.ISupportInitialize)(this.pictureBox9
                    )).BeginInit();
129             ((System.ComponentModel.ISupportInitialize)(this.pictureBox8
                    )).BeginInit();
```

```
130             ((System.ComponentModel.ISupportInitialize)(this.pictureBox7
                    )).BeginInit();
131             ((System.ComponentModel.ISupportInitialize)(this.pictureBox6
                    )).BeginInit();
132             ((System.ComponentModel.ISupportInitialize)(this.pictureBox5
                    )).BeginInit();
133             ((System.ComponentModel.ISupportInitialize)(this.pictureBox4
                    )).BeginInit();
134             ((System.ComponentModel.ISupportInitialize)(this.pictureBox3
                    )).BeginInit();
135             ((System.ComponentModel.ISupportInitialize)(this.pictureBox2
                    )).BeginInit();
136             ((System.ComponentModel.ISupportInitialize)(this.pictureBox1
                    )).BeginInit();
137             this.groupBox11.SuspendLayout();
138             this.menuStrip1.SuspendLayout();
139             this.groupBox12.SuspendLayout();
140             ((System.ComponentModel.ISupportInitialize)(this.
                    NumericGenMem)).BeginInit();
141             this.SuspendLayout();
142             //
143             // PBAutomataSimulator
144             //
145             this.PBAutomataSimulator.BackColor = System.Drawing.
                    SystemColors.ActiveCaptionText;
146             this.PBAutomataSimulator.Location = new System.Drawing.Point
                    (3, 3);
147             this.PBAutomataSimulator.Name = "PBAutomataSimulator";
148             this.PBAutomataSimulator.Size = new System.Drawing.Size(545,
                     505);
149             this.PBAutomataSimulator.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.AutoSize;
150             this.PBAutomataSimulator.TabIndex = 1;
151             this.PBAutomataSimulator.TabStop = false;
152             this.PBAutomataSimulator.Paint += new System.Windows.Forms.
                    PaintEventHandler(this.PBAutomataSimulator_Paint);
153             this.PBAutomataSimulator.MouseDown += new System.Windows.
                    Forms.MouseEventHandler(this.
                    PBAutomataSimulator_MouseDown);
154             this.PBAutomataSimulator.MouseMove += new System.Windows.
                    Forms.MouseEventHandler(this.
                    PBAutomataSimulator_MouseMove);
155             //
156             // CHHistogram
157             //
158             chartArea3.Name = "ChartArea1";
159             this.CHHistogram.ChartAreas.Add(chartArea3);
160             legend3.Name = "Legend1";
161             this.CHHistogram.Legends.Add(legend3);
162             this.CHHistogram.Location = new System.Drawing.Point(6, 19);
163             this.CHHistogram.Name = "CHHistogram";
164             series3.ChartArea = "ChartArea1";
165             series3.Legend = "Legend1";
```

```
166             series3.Name = "#Ones";
167             this.CHHistogram.Series.Add(series3);
168             this.CHHistogram.Size = new System.Drawing.Size(584, 337);
169             this.CHHistogram.TabIndex = 2;
170             this.CHHistogram.Text = "chart1";
171             //
172             // groupBox1
173             //
174             this.groupBox1.Controls.Add(this.CHHistogram);
175             this.groupBox1.Location = new System.Drawing.Point(13, 31);
176             this.groupBox1.Name = "groupBox1";
177             this.groupBox1.Size = new System.Drawing.Size(596, 362);
178             this.groupBox1.TabIndex = 3;
179             this.groupBox1.TabStop = false;
180             this.groupBox1.Text = "Histogram";
181             //
182             // BTNStart
183             //
184             this.BTNStart.Location = new System.Drawing.Point(6, 19);
185             this.BTNStart.Name = "BTNStart";
186             this.BTNStart.Size = new System.Drawing.Size(75, 23);
187             this.BTNStart.TabIndex = 4;
188             this.BTNStart.Text = "Start";
189             this.BTNStart.UseVisualStyleBackColor = true;
190             this.BTNStart.Click += new System.EventHandler(this.
                    BTNStart_Click);
191             //
192             // groupBox2
193             //
194             this.groupBox2.Controls.Add(this.label5);
195             this.groupBox2.Controls.Add(this.label4);
196             this.groupBox2.Controls.Add(this.label3);
197             this.groupBox2.Controls.Add(this.label2);
198             this.groupBox2.Controls.Add(this.ComboBY2i);
199             this.groupBox2.Controls.Add(this.ComboBX2i);
200             this.groupBox2.Controls.Add(this.ComboBYi);
201             this.groupBox2.Controls.Add(this.ComboBXi);
202             this.groupBox2.Location = new System.Drawing.Point(13, 402);
203             this.groupBox2.Name = "groupBox2";
204             this.groupBox2.Size = new System.Drawing.Size(107, 137);
205             this.groupBox2.TabIndex = 6;
206             this.groupBox2.TabStop = false;
207             this.groupBox2.Text = "Rules";
208             //
209             // label5
210             //
211             this.label5.AutoSize = true;
212             this.label5.Location = new System.Drawing.Point(11, 111);
213             this.label5.Name = "label5";
214             this.label5.Size = new System.Drawing.Size(26, 13);
215             this.label5.TabIndex = 7;
216             this.label5.Text = "Y_2";
217             //
```

```
218             // label4
219             //
220             this.label4.AutoSize = true;
221             this.label4.Location = new System.Drawing.Point(11, 83);
222             this.label4.Name = "label4";
223             this.label4.Size = new System.Drawing.Size(26, 13);
224             this.label4.TabIndex = 6;
225             this.label4.Text = "X_2";
226             //
227             // label3
228             //
229             this.label3.AutoSize = true;
230             this.label3.Location = new System.Drawing.Point(10, 55);
231             this.label3.Name = "label3";
232             this.label3.Size = new System.Drawing.Size(14, 13);
233             this.label3.TabIndex = 5;
234             this.label3.Text = "Y";
235             //
236             // label2
237             //
238             this.label2.AutoSize = true;
239             this.label2.Location = new System.Drawing.Point(8, 27);
240             this.label2.Name = "label2";
241             this.label2.Size = new System.Drawing.Size(14, 13);
242             this.label2.TabIndex = 4;
243             this.label2.Text = "X";
244             //
245             // ComboBY2i
246             //
247             this.ComboBY2i.FormattingEnabled = true;
248             this.ComboBY2i.Items.AddRange(new object[] {
249             "1",
250             "2",
251             "3",
252             "4",
253             "5",
254             "6",
255             "7",
256             "8"});
257             this.ComboBY2i.Location = new System.Drawing.Point(47, 104);
258             this.ComboBY2i.Name = "ComboBY2i";
259             this.ComboBY2i.Size = new System.Drawing.Size(43, 21);
260             this.ComboBY2i.TabIndex = 3;
261             this.ComboBY2i.Text = "3";
262             //
263             // ComboBX2i
264             //
265             this.ComboBX2i.FormattingEnabled = true;
266             this.ComboBX2i.Items.AddRange(new object[] {
267             "1",
268             "2",
269             "3",
270             "4",
```

```
271             "5",
272             "6",
273             "7",
274             "8"});
275             this.ComboBX2i.Location = new System.Drawing.Point(47, 76);
276             this.ComboBX2i.Name = "ComboBX2i";
277             this.ComboBX2i.Size = new System.Drawing.Size(43, 21);
278             this.ComboBX2i.TabIndex = 2;
279             this.ComboBX2i.Text = "3";
280             //
281             // ComboBYi
282             //
283             this.ComboBYi.FormattingEnabled = true;
284             this.ComboBYi.Items.AddRange(new object[] {
285             "1",
286             "2",
287             "3",
288             "4",
289             "5",
290             "6",
291             "7",
292             "8"});
293             this.ComboBYi.Location = new System.Drawing.Point(48, 48);
294             this.ComboBYi.Name = "ComboBYi";
295             this.ComboBYi.Size = new System.Drawing.Size(43, 21);
296             this.ComboBYi.TabIndex = 1;
297             this.ComboBYi.Text = "3";
298             //
299             // ComboBXi
300             //
301             this.ComboBXi.FormattingEnabled = true;
302             this.ComboBXi.Items.AddRange(new object[] {
303             "1",
304             "2",
305             "3",
306             "4",
307             "5",
308             "6",
309             "7",
310             "8"});
311             this.ComboBXi.Location = new System.Drawing.Point(48, 20);
312             this.ComboBXi.Name = "ComboBXi";
313             this.ComboBXi.Size = new System.Drawing.Size(43, 21);
314             this.ComboBXi.TabIndex = 0;
315             this.ComboBXi.Text = "2";
316             //
317             // TXTGeneration
318             //
319             this.TXTGeneration.AutoSize = true;
320             this.TXTGeneration.BackColor = System.Drawing.Color.
                    Transparent;
321             this.TXTGeneration.ForeColor = System.Drawing.Color.Black;
322             this.TXTGeneration.Location = new System.Drawing.Point(6,
```

```
            22);
323         this.TXTGeneration.Name = "TXTGeneration";
324         this.TXTGeneration.Size = new System.Drawing.Size(62, 13);
325         this.TXTGeneration.TabIndex = 7;
326         this.TXTGeneration.Text = "Generation ";
327         //
328         // TXTPopulation
329         //
330         this.TXTPopulation.AutoSize = true;
331         this.TXTPopulation.Location = new System.Drawing.Point(6,
                44);
332         this.TXTPopulation.Name = "TXTPopulation";
333         this.TXTPopulation.Size = new System.Drawing.Size(57, 13);
334         this.TXTPopulation.TabIndex = 8;
335         this.TXTPopulation.Text = "Population";
336         //
337         // BTNStep
338         //
339         this.BTNStep.Location = new System.Drawing.Point(87, 19);
340         this.BTNStep.Name = "BTNStep";
341         this.BTNStep.Size = new System.Drawing.Size(75, 23);
342         this.BTNStep.TabIndex = 9;
343         this.BTNStep.Text = "One step";
344         this.BTNStep.UseVisualStyleBackColor = true;
345         this.BTNStep.Click += new System.EventHandler(this.
                BTNStep_Click);
346         //
347         // TBSpeed
348         //
349         this.TBSpeed.Location = new System.Drawing.Point(6, 19);
350         this.TBSpeed.Maximum = 3000;
351         this.TBSpeed.Minimum = 10;
352         this.TBSpeed.Name = "TBSpeed";
353         this.TBSpeed.Size = new System.Drawing.Size(233, 45);
354         this.TBSpeed.TabIndex = 10;
355         this.TBSpeed.Value = 1000;
356         this.TBSpeed.ValueChanged += new System.EventHandler(this.
                trackBar1_ValueChanged);
357         //
358         // TimerSimulation
359         //
360         this.TimerSimulation.Tick += new System.EventHandler(this.
                TimerSimulation_Tick);
361         //
362         // flowLayoutPanel1
363         //
364         this.flowLayoutPanel1.Anchor = ((System.Windows.Forms.
                AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
                 System.Windows.Forms.AnchorStyles.Bottom)
365         | System.Windows.Forms.AnchorStyles.Left)
366         | System.Windows.Forms.AnchorStyles.Right)));
367         this.flowLayoutPanel1.AutoScroll = true;
368         this.flowLayoutPanel1.AutoSizeMode = System.Windows.Forms.
```

```
            AutoSizeMode.GrowAndShrink;
369         this.flowLayoutPanel1.Controls.Add(this.PBAutomataSimulator)
            ;
370         this.flowLayoutPanel1.Location = new System.Drawing.Point
            (618, 100);
371         this.flowLayoutPanel1.MinimumSize = new System.Drawing.Size
            (639, 600);
372         this.flowLayoutPanel1.Name = "flowLayoutPanel1";
373         this.flowLayoutPanel1.Size = new System.Drawing.Size(639,
            600);
374         this.flowLayoutPanel1.TabIndex = 11;
375         //
376         // groupBox3
377         //
378         this.groupBox3.Controls.Add(this.label6);
379         this.groupBox3.Controls.Add(this.label1);
380         this.groupBox3.Controls.Add(this.TBSpeed);
381         this.groupBox3.Location = new System.Drawing.Point(16, 631);
382         this.groupBox3.Name = "groupBox3";
383         this.groupBox3.Size = new System.Drawing.Size(242, 69);
384         this.groupBox3.TabIndex = 12;
385         this.groupBox3.TabStop = false;
386         this.groupBox3.Text = "Speed";
387         //
388         // label6
389         //
390         this.label6.AutoSize = true;
391         this.label6.Location = new System.Drawing.Point(217, 53);
392         this.label6.Name = "label6";
393         this.label6.Size = new System.Drawing.Size(18, 13);
394         this.label6.TabIndex = 12;
395         this.label6.Text = "3s";
396         //
397         // label1
398         //
399         this.label1.AutoSize = true;
400         this.label1.Location = new System.Drawing.Point(6, 51);
401         this.label1.Name = "label1";
402         this.label1.Size = new System.Drawing.Size(32, 13);
403         this.label1.TabIndex = 11;
404         this.label1.Text = "10ms";
405         //
406         // BTNZoomP
407         //
408         this.BTNZoomP.Location = new System.Drawing.Point(534, 424);
409         this.BTNZoomP.Name = "BTNZoomP";
410         this.BTNZoomP.Size = new System.Drawing.Size(75, 23);
411         this.BTNZoomP.TabIndex = 13;
412         this.BTNZoomP.Text = "zoom +";
413         this.BTNZoomP.UseVisualStyleBackColor = true;
414         this.BTNZoomP.Click += new System.EventHandler(this.
            BTNZoomP_Click);
415         //
```

```
416             // BTNZoomM
417             //
418             this.BTNZoomM.Location = new System.Drawing.Point(534, 457);
419             this.BTNZoomM.Name = "BTNZoomM";
420             this.BTNZoomM.Size = new System.Drawing.Size(75, 23);
421             this.BTNZoomM.TabIndex = 14;
422             this.BTNZoomM.Text = "zoom -";
423             this.BTNZoomM.UseVisualStyleBackColor = true;
424             this.BTNZoomM.Click += new System.EventHandler(this.
                    BTNZoomM_Click);
425             //
426             // groupBox4
427             //
428             this.groupBox4.Controls.Add(this.CBPatternRecognition);
429             this.groupBox4.Controls.Add(this.BTNStart);
430             this.groupBox4.Controls.Add(this.BTNStep);
431             this.groupBox4.Location = new System.Drawing.Point(128, 402)
                    ;
432             this.groupBox4.Name = "groupBox4";
433             this.groupBox4.Size = new System.Drawing.Size(168, 69);
434             this.groupBox4.TabIndex = 15;
435             this.groupBox4.TabStop = false;
436             this.groupBox4.Text = "Controlls";
437             //
438             // CBPatternRecognition
439             //
440             this.CBPatternRecognition.AutoSize = true;
441             this.CBPatternRecognition.Location = new System.Drawing.
                    Point(24, 48);
442             this.CBPatternRecognition.Name = "CBPatternRecognition";
443             this.CBPatternRecognition.Size = new System.Drawing.Size
                    (115, 17);
444             this.CBPatternRecognition.TabIndex = 29;
445             this.CBPatternRecognition.Text = "Pattern recognition";
446             this.CBPatternRecognition.UseVisualStyleBackColor = true;
447             //
448             // groupBox5
449             //
450             this.groupBox5.Controls.Add(this.button1);
451             this.groupBox5.Controls.Add(this.numericOnes);
452             this.groupBox5.Controls.Add(this.label7);
453             this.groupBox5.Location = new System.Drawing.Point(166, 471)
                    ;
454             this.groupBox5.Name = "groupBox5";
455             this.groupBox5.Size = new System.Drawing.Size(92, 82);
456             this.groupBox5.TabIndex = 16;
457             this.groupBox5.TabStop = false;
458             this.groupBox5.Text = "Random";
459             //
460             // button1
461             //
462             this.button1.Location = new System.Drawing.Point(9, 55);
463             this.button1.Name = "button1";
```

```
464             this.button1.Size = new System.Drawing.Size(75, 23);
465             this.button1.TabIndex = 4;
466             this.button1.Text = "Generate";
467             this.button1.UseVisualStyleBackColor = true;
468             this.button1.Click += new System.EventHandler(this.
                    button1_Click);
469             //
470             // numericOnes
471             //
472             this.numericOnes.DecimalPlaces = 3;
473             this.numericOnes.Location = new System.Drawing.Point(9, 32);
474             this.numericOnes.Name = "numericOnes";
475             this.numericOnes.Size = new System.Drawing.Size(76, 20);
476             this.numericOnes.TabIndex = 2;
477             //
478             // label7
479             //
480             this.label7.AutoSize = true;
481             this.label7.Location = new System.Drawing.Point(33, 17);
482             this.label7.Name = "label7";
483             this.label7.Size = new System.Drawing.Size(18, 13);
484             this.label7.TabIndex = 0;
485             this.label7.Text = "1s";
486             //
487             // groupBox6
488             //
489             this.groupBox6.Controls.Add(this.BTNGrid);
490             this.groupBox6.Controls.Add(this.BTNDeadCells);
491             this.groupBox6.Controls.Add(this.BTNAliveCells);
492             this.groupBox6.Location = new System.Drawing.Point(306, 404)
                    ;
493             this.groupBox6.Name = "groupBox6";
494             this.groupBox6.Size = new System.Drawing.Size(106, 157);
495             this.groupBox6.TabIndex = 17;
496             this.groupBox6.TabStop = false;
497             this.groupBox6.Text = "Choose colors";
498             //
499             // BTNGrid
500             //
501             this.BTNGrid.Location = new System.Drawing.Point(15, 119);
502             this.BTNGrid.Name = "BTNGrid";
503             this.BTNGrid.Size = new System.Drawing.Size(75, 23);
504             this.BTNGrid.TabIndex = 7;
505             this.BTNGrid.Text = "Grid";
506             this.BTNGrid.UseVisualStyleBackColor = true;
507             this.BTNGrid.Click += new System.EventHandler(this.
                    BTNGrid_Click);
508             //
509             // BTNDeadCells
510             //
511             this.BTNDeadCells.Location = new System.Drawing.Point(15,
                    74);
512             this.BTNDeadCells.Name = "BTNDeadCells";
```

```
513             this.BTNDeadCells.Size = new System.Drawing.Size(75, 23);
514             this.BTNDeadCells.TabIndex = 6;
515             this.BTNDeadCells.Text = "Dead cells";
516             this.BTNDeadCells.UseVisualStyleBackColor = true;
517             this.BTNDeadCells.Click += new System.EventHandler(this.
                    BTNDeadCells_Click);
518             //
519             // BTNAliveCells
520             //
521             this.BTNAliveCells.Location = new System.Drawing.Point(15,
                    28);
522             this.BTNAliveCells.Name = "BTNAliveCells";
523             this.BTNAliveCells.Size = new System.Drawing.Size(75, 23);
524             this.BTNAliveCells.TabIndex = 5;
525             this.BTNAliveCells.Text = "Alive cells";
526             this.BTNAliveCells.UseVisualStyleBackColor = true;
527             this.BTNAliveCells.Click += new System.EventHandler(this.
                    button2_Click_1);
528             //
529             // BTNSelectFile
530             //
531             this.BTNSelectFile.Location = new System.Drawing.Point(3,
                    17);
532             this.BTNSelectFile.Name = "BTNSelectFile";
533             this.BTNSelectFile.Size = new System.Drawing.Size(75, 23);
534             this.BTNSelectFile.TabIndex = 18;
535             this.BTNSelectFile.Text = "Choose file";
536             this.BTNSelectFile.UseVisualStyleBackColor = true;
537             this.BTNSelectFile.Click += new System.EventHandler(this.
                    button2_Click);
538             //
539             // BTNClear
540             //
541             this.BTNClear.Location = new System.Drawing.Point(534, 489);
542             this.BTNClear.Name = "BTNClear";
543             this.BTNClear.Size = new System.Drawing.Size(75, 23);
544             this.BTNClear.TabIndex = 19;
545             this.BTNClear.Text = "Clear";
546             this.BTNClear.UseVisualStyleBackColor = true;
547             this.BTNClear.Click += new System.EventHandler(this.
                    BTNClear_Click);
548             //
549             // groupBox7
550             //
551             this.groupBox7.Controls.Add(this.BTNCreateMatrix);
552             this.groupBox7.Controls.Add(this.numericCols);
553             this.groupBox7.Controls.Add(this.numericRows);
554             this.groupBox7.Controls.Add(this.label11);
555             this.groupBox7.Controls.Add(this.label8);
556             this.groupBox7.Location = new System.Drawing.Point(418, 404)
                    ;
557             this.groupBox7.Name = "groupBox7";
558             this.groupBox7.Size = new System.Drawing.Size(104, 157);
```

```
559             this.groupBox7.TabIndex = 20;
560             this.groupBox7.TabStop = false;
561             this.groupBox7.Text = "Size";
562             //
563             // BTNCreateMatrix
564             //
565             this.BTNCreateMatrix.Location = new System.Drawing.Point(13,
                    119);
566             this.BTNCreateMatrix.Name = "BTNCreateMatrix";
567             this.BTNCreateMatrix.Size = new System.Drawing.Size(81, 23);
568             this.BTNCreateMatrix.TabIndex = 21;
569             this.BTNCreateMatrix.Text = "Create";
570             this.BTNCreateMatrix.UseVisualStyleBackColor = true;
571             this.BTNCreateMatrix.Click += new System.EventHandler(this.
                    BTNCreateMatrix_Click);
572             //
573             // numericCols
574             //
575             this.numericCols.Location = new System.Drawing.Point(13, 82)
                    ;
576             this.numericCols.Maximum = new decimal(new int[] {
577             1000,
578             0,
579             0,
580             0});
581             this.numericCols.Minimum = new decimal(new int[] {
582             10,
583             0,
584             0,
585             0});
586             this.numericCols.Name = "numericCols";
587             this.numericCols.Size = new System.Drawing.Size(81, 20);
588             this.numericCols.TabIndex = 3;
589             this.numericCols.Value = new decimal(new int[] {
590             10,
591             0,
592             0,
593             0});
594             //
595             // numericRows
596             //
597             this.numericRows.Location = new System.Drawing.Point(11, 42)
                    ;
598             this.numericRows.Maximum = new decimal(new int[] {
599             1000,
600             0,
601             0,
602             0});
603             this.numericRows.Minimum = new decimal(new int[] {
604             10,
605             0,
606             0,
607             0});
```

```
608             this.numericRows.Name = "numericRows";
609             this.numericRows.Size = new System.Drawing.Size(81, 20);
610             this.numericRows.TabIndex = 2;
611             this.numericRows.Value = new decimal(new int[] {
612             10,
613             0,
614             0,
615             0});
616             //
617             // label11
618             //
619             this.label11.AutoSize = true;
620             this.label11.Location = new System.Drawing.Point(36, 66);
621             this.label11.Name = "label11";
622             this.label11.Size = new System.Drawing.Size(27, 13);
623             this.label11.TabIndex = 1;
624             this.label11.Text = "Cols";
625             //
626             // label8
627             //
628             this.label8.AutoSize = true;
629             this.label8.Location = new System.Drawing.Point(29, 25);
630             this.label8.Name = "label8";
631             this.label8.Size = new System.Drawing.Size(34, 13);
632             this.label8.TabIndex = 0;
633             this.label8.Text = "Rows";
634             //
635             // groupBox8
636             //
637             this.groupBox8.Controls.Add(this.BTNSavePatterns);
638             this.groupBox8.Controls.Add(this.BTNSave);
639             this.groupBox8.Controls.Add(this.BTNSelectFile);
640             this.groupBox8.Location = new System.Drawing.Point(125, 557)
                    ;
641             this.groupBox8.Name = "groupBox8";
642             this.groupBox8.Size = new System.Drawing.Size(171, 75);
643             this.groupBox8.TabIndex = 21;
644             this.groupBox8.TabStop = false;
645             this.groupBox8.Text = "File options";
646             //
647             // BTNSavePatterns
648             //
649             this.BTNSavePatterns.Location = new System.Drawing.Point(36,
                     46);
650             this.BTNSavePatterns.Name = "BTNSavePatterns";
651             this.BTNSavePatterns.Size = new System.Drawing.Size(96, 23);
652             this.BTNSavePatterns.TabIndex = 19;
653             this.BTNSavePatterns.Text = "Save patterns";
654             this.BTNSavePatterns.UseVisualStyleBackColor = true;
655             this.BTNSavePatterns.Click += new System.EventHandler(this.
                    BTNSavePatterns_Click);
656             //
657             // BTNSave
```

```
658             //
659             this.BTNSave.Location = new System.Drawing.Point(90, 17);
660             this.BTNSave.Name = "BTNSave";
661             this.BTNSave.Size = new System.Drawing.Size(75, 23);
662             this.BTNSave.TabIndex = 0;
663             this.BTNSave.Text = "Save states";
664             this.BTNSave.UseVisualStyleBackColor = true;
665             this.BTNSave.Click += new System.EventHandler(this.
                    BTNSave_Click);
666             //
667             // label12
668             //
669             this.label12.AutoSize = true;
670             this.label12.Location = new System.Drawing.Point(6, 66);
671             this.label12.Name = "label12";
672             this.label12.Size = new System.Drawing.Size(83, 13);
673             this.label12.TabIndex = 22;
674             this.label12.Text = "Total population";
675             //
676             // label13
677             //
678             this.label13.AutoSize = true;
679             this.label13.Location = new System.Drawing.Point(6, 88);
680             this.label13.Name = "label13";
681             this.label13.Size = new System.Drawing.Size(47, 13);
682             this.label13.TabIndex = 23;
683             this.label13.Text = "Average";
684             //
685             // label14
686             //
687             this.label14.AutoSize = true;
688             this.label14.Location = new System.Drawing.Point(6, 110);
689             this.label14.Name = "label14";
690             this.label14.Size = new System.Drawing.Size(42, 13);
691             this.label14.TabIndex = 24;
692             this.label14.Text = "Density";
693             //
694             // groupBox9
695             //
696             this.groupBox9.Controls.Add(this.CBPoints);
697             this.groupBox9.Controls.Add(this.CheckGraphEnabled);
698             this.groupBox9.Location = new System.Drawing.Point(15, 545);
699             this.groupBox9.Name = "groupBox9";
700             this.groupBox9.Size = new System.Drawing.Size(104, 72);
701             this.groupBox9.TabIndex = 25;
702             this.groupBox9.TabStop = false;
703             this.groupBox9.Text = "Graph options";
704             //
705             // CBPoints
706             //
707             this.CBPoints.AutoSize = true;
708             this.CBPoints.Location = new System.Drawing.Point(16, 41);
709             this.CBPoints.Name = "CBPoints";
```

```
710             this.CBPoints.Size = new System.Drawing.Size(55, 17);
711             this.CBPoints.TabIndex = 1;
712             this.CBPoints.Text = "Points";
713             this.CBPoints.UseVisualStyleBackColor = true;
714             this.CBPoints.CheckedChanged += new System.EventHandler(this
                    .CBPoints_CheckedChanged);
715             //
716             // CheckGraphEnabled
717             //
718             this.CheckGraphEnabled.AutoSize = true;
719             this.CheckGraphEnabled.Checked = true;
720             this.CheckGraphEnabled.CheckState = System.Windows.Forms.
                    CheckState.Checked;
721             this.CheckGraphEnabled.Location = new System.Drawing.Point
                    (16, 23);
722             this.CheckGraphEnabled.Name = "CheckGraphEnabled";
723             this.CheckGraphEnabled.Size = new System.Drawing.Size(59,
                    17);
724             this.CheckGraphEnabled.TabIndex = 0;
725             this.CheckGraphEnabled.Text = "Enable";
726             this.CheckGraphEnabled.UseVisualStyleBackColor = true;
727             //
728             // groupBox10
729             //
730             this.groupBox10.Controls.Add(this.TXTGeneration);
731             this.groupBox10.Controls.Add(this.TXTPopulation);
732             this.groupBox10.Controls.Add(this.label14);
733             this.groupBox10.Controls.Add(this.label12);
734             this.groupBox10.Controls.Add(this.label13);
735             this.groupBox10.Location = new System.Drawing.Point(418,
                    567);
736             this.groupBox10.Name = "groupBox10";
737             this.groupBox10.Size = new System.Drawing.Size(191, 133);
738             this.groupBox10.TabIndex = 26;
739             this.groupBox10.TabStop = false;
740             this.groupBox10.Text = "Data";
741             //
742             // pictureBox10
743             //
744             this.pictureBox10.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
745             this.pictureBox10.Cursor = System.Windows.Forms.Cursors.Hand
                    ;
746             this.pictureBox10.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox10.Image")));
747             this.pictureBox10.Location = new System.Drawing.Point(581,
                    16);
748             this.pictureBox10.Name = "pictureBox10";
749             this.pictureBox10.Size = new System.Drawing.Size(38, 50);
750             this.pictureBox10.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
751             this.pictureBox10.TabIndex = 9;
752             this.pictureBox10.TabStop = false;
```

```
753             this.pictureBox10.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox10_MouseUp);
754             //
755             // pictureBox9
756             //
757             this.pictureBox9.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
758             this.pictureBox9.Cursor = System.Windows.Forms.Cursors.Hand;
759             this.pictureBox9.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox9.Image")));
760             this.pictureBox9.Location = new System.Drawing.Point(512,
                    31);
761             this.pictureBox9.Name = "pictureBox9";
762             this.pictureBox9.Size = new System.Drawing.Size(63, 26);
763             this.pictureBox9.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
764             this.pictureBox9.TabIndex = 8;
765             this.pictureBox9.TabStop = false;
766             this.pictureBox9.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox9_MouseUp);
767             //
768             // pictureBox8
769             //
770             this.pictureBox8.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
771             this.pictureBox8.Cursor = System.Windows.Forms.Cursors.Hand;
772             this.pictureBox8.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox8.Image")));
773             this.pictureBox8.Location = new System.Drawing.Point(456,
                    16);
774             this.pictureBox8.Name = "pictureBox8";
775             this.pictureBox8.Size = new System.Drawing.Size(50, 50);
776             this.pictureBox8.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
777             this.pictureBox8.TabIndex = 7;
778             this.pictureBox8.TabStop = false;
779             this.pictureBox8.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox8_MouseUp);
780             //
781             // pictureBox7
782             //
783             this.pictureBox7.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
784             this.pictureBox7.Cursor = System.Windows.Forms.Cursors.Hand;
785             this.pictureBox7.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox7.Image")));
786             this.pictureBox7.Location = new System.Drawing.Point(412,
                    16);
787             this.pictureBox7.Name = "pictureBox7";
788             this.pictureBox7.Size = new System.Drawing.Size(38, 50);
789             this.pictureBox7.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
790             this.pictureBox7.TabIndex = 6;
```

```
791             this.pictureBox7.TabStop = false;
792             this.pictureBox7.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox7_MouseUp);
793             //
794             // pictureBox6
795             //
796             this.pictureBox6.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
797             this.pictureBox6.Cursor = System.Windows.Forms.Cursors.Hand;
798             this.pictureBox6.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox6.Image")));
799             this.pictureBox6.Location = new System.Drawing.Point(329,
                    16);
800             this.pictureBox6.Name = "pictureBox6";
801             this.pictureBox6.Size = new System.Drawing.Size(77, 50);
802             this.pictureBox6.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
803             this.pictureBox6.TabIndex = 5;
804             this.pictureBox6.TabStop = false;
805             this.pictureBox6.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox6_MouseUp);
806             //
807             // pictureBox5
808             //
809             this.pictureBox5.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
810             this.pictureBox5.Cursor = System.Windows.Forms.Cursors.Hand;
811             this.pictureBox5.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox5.Image")));
812             this.pictureBox5.Location = new System.Drawing.Point(273,
                    16);
813             this.pictureBox5.Name = "pictureBox5";
814             this.pictureBox5.Size = new System.Drawing.Size(50, 50);
815             this.pictureBox5.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
816             this.pictureBox5.TabIndex = 4;
817             this.pictureBox5.TabStop = false;
818             this.pictureBox5.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox5_MouseUp);
819             //
820             // pictureBox4
821             //
822             this.pictureBox4.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
823             this.pictureBox4.Cursor = System.Windows.Forms.Cursors.Hand;
824             this.pictureBox4.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox4.Image")));
825             this.pictureBox4.Location = new System.Drawing.Point(217,
                    16);
826             this.pictureBox4.Name = "pictureBox4";
827             this.pictureBox4.Size = new System.Drawing.Size(50, 50);
828             this.pictureBox4.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
```

```
829             this.pictureBox4.TabIndex = 3;
830             this.pictureBox4.TabStop = false;
831             this.pictureBox4.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox4_MouseUp);
832             //
833             // pictureBox3
834             //
835             this.pictureBox3.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
836             this.pictureBox3.Cursor = System.Windows.Forms.Cursors.Hand;
837             this.pictureBox3.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox3.Image")));
838             this.pictureBox3.Location = new System.Drawing.Point(161,
                    16);
839             this.pictureBox3.Name = "pictureBox3";
840             this.pictureBox3.Size = new System.Drawing.Size(50, 50);
841             this.pictureBox3.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
842             this.pictureBox3.TabIndex = 2;
843             this.pictureBox3.TabStop = false;
844             this.pictureBox3.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox3_MouseUp);
845             //
846             // pictureBox2
847             //
848             this.pictureBox2.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
849             this.pictureBox2.Cursor = System.Windows.Forms.Cursors.Hand;
850             this.pictureBox2.ErrorImage = ((System.Drawing.Image)(
                    resources.GetObject("pictureBox2.ErrorImage")));
851             this.pictureBox2.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox2.Image")));
852             this.pictureBox2.Location = new System.Drawing.Point(70, 16)
                    ;
853             this.pictureBox2.Name = "pictureBox2";
854             this.pictureBox2.Size = new System.Drawing.Size(85, 50);
855             this.pictureBox2.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
856             this.pictureBox2.TabIndex = 1;
857             this.pictureBox2.TabStop = false;
858             this.pictureBox2.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox2_MouseUp);
859             //
860             // pictureBox1
861             //
862             this.pictureBox1.BorderStyle = System.Windows.Forms.
                    BorderStyle.FixedSingle;
863             this.pictureBox1.Cursor = System.Windows.Forms.Cursors.Hand;
864             this.pictureBox1.Image = ((System.Drawing.Image)(resources.
                    GetObject("pictureBox1.Image")));
865             this.pictureBox1.Location = new System.Drawing.Point(14, 16)
                    ;
866             this.pictureBox1.Name = "pictureBox1";
```

```
867             this.pictureBox1.Size = new System.Drawing.Size(50, 50);
868             this.pictureBox1.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.StretchImage;
869             this.pictureBox1.TabIndex = 0;
870             this.pictureBox1.TabStop = false;
871             this.pictureBox1.MouseUp += new System.Windows.Forms.
                    MouseEventHandler(this.pictureBox1_MouseUp);
872             //
873             // groupBox11
874             //
875             this.groupBox11.Controls.Add(this.pictureBox1);
876             this.groupBox11.Controls.Add(this.pictureBox2);
877             this.groupBox11.Controls.Add(this.pictureBox10);
878             this.groupBox11.Controls.Add(this.pictureBox3);
879             this.groupBox11.Controls.Add(this.pictureBox5);
880             this.groupBox11.Controls.Add(this.pictureBox4);
881             this.groupBox11.Controls.Add(this.pictureBox9);
882             this.groupBox11.Controls.Add(this.pictureBox6);
883             this.groupBox11.Controls.Add(this.pictureBox7);
884             this.groupBox11.Controls.Add(this.pictureBox8);
885             this.groupBox11.Location = new System.Drawing.Point(621, 23)
                    ;
886             this.groupBox11.Name = "groupBox11";
887             this.groupBox11.Size = new System.Drawing.Size(633, 74);
888             this.groupBox11.TabIndex = 27;
889             this.groupBox11.TabStop = false;
890             this.groupBox11.Text = "Patterns";
891             //
892             // menuStrip1
893             //
894             this.menuStrip1.Items.AddRange(new System.Windows.Forms.
                    ToolStripItem[] {
895             this.generarPatronesToolStripMenuItem});
896             this.menuStrip1.Location = new System.Drawing.Point(0, 0);
897             this.menuStrip1.Name = "menuStrip1";
898             this.menuStrip1.Size = new System.Drawing.Size(1266, 24);
899             this.menuStrip1.TabIndex = 28;
900             this.menuStrip1.Text = "menuStrip1";
901             //
902             // generarPatronesToolStripMenuItem
903             //
904             this.generarPatronesToolStripMenuItem.DropDownItems.AddRange
                    (new System.Windows.Forms.ToolStripItem[] {
905             this.generatePatternsToolStripMenuItem});
906             this.generarPatronesToolStripMenuItem.Name = "
                    generarPatronesToolStripMenuItem";
907             this.generarPatronesToolStripMenuItem.Size = new System.
                    Drawing.Size(37, 20);
908             this.generarPatronesToolStripMenuItem.Text = "File";
909             //
910             // generatePatternsToolStripMenuItem
911             //
912             this.generatePatternsToolStripMenuItem.Name = "
```

```
            generatePatternsToolStripMenuItem";
913         this.generatePatternsToolStripMenuItem.Size = new System.
                Drawing.Size(167, 22);
914         this.generatePatternsToolStripMenuItem.Text = "Generate
                patterns";
915         this.generatePatternsToolStripMenuItem.Click += new System.
                EventHandler(this.
                generatePatternsToolStripMenuItem_Click);
916         //
917         // groupBox12
918         //
919         this.groupBox12.Controls.Add(this.label10);
920         this.groupBox12.Controls.Add(this.label9);
921         this.groupBox12.Controls.Add(this.comboRuleMem);
922         this.groupBox12.Controls.Add(this.NumericGenMem);
923         this.groupBox12.Location = new System.Drawing.Point(306,
                574);
924         this.groupBox12.Name = "groupBox12";
925         this.groupBox12.Size = new System.Drawing.Size(106, 123);
926         this.groupBox12.TabIndex = 29;
927         this.groupBox12.TabStop = false;
928         this.groupBox12.Text = "Memory";
929         //
930         // label10
931         //
932         this.label10.AutoSize = true;
933         this.label10.Location = new System.Drawing.Point(18, 67);
934         this.label10.Name = "label10";
935         this.label10.Size = new System.Drawing.Size(29, 13);
936         this.label10.TabIndex = 3;
937         this.label10.Text = "Rule";
938         //
939         // label9
940         //
941         this.label9.AutoSize = true;
942         this.label9.Location = new System.Drawing.Point(13, 21);
943         this.label9.Name = "label9";
944         this.label9.Size = new System.Drawing.Size(79, 13);
945         this.label9.TabIndex = 2;
946         this.label9.Text = "Generations (T)";
947         //
948         // comboRuleMem
949         //
950         this.comboRuleMem.FormattingEnabled = true;
951         this.comboRuleMem.Items.AddRange(new object[] {
952         "Majority",
953         "Minority",
954         "Parity"});
955         this.comboRuleMem.Location = new System.Drawing.Point(15,
                88);
956         this.comboRuleMem.Name = "comboRuleMem";
957         this.comboRuleMem.Size = new System.Drawing.Size(75, 21);
958         this.comboRuleMem.TabIndex = 1;
```

```
959             this.comboRuleMem.Text = "Majority";
960             //
961             // NumericGenMem
962             //
963             this.NumericGenMem.Location = new System.Drawing.Point(13,
                    39);
964             this.NumericGenMem.Minimum = new decimal(new int[] {
965             3,
966             0,
967             0,
968             0});
969             this.NumericGenMem.Name = "NumericGenMem";
970             this.NumericGenMem.Size = new System.Drawing.Size(77, 20);
971             this.NumericGenMem.TabIndex = 0;
972             this.NumericGenMem.Value = new decimal(new int[] {
973             3,
974             0,
975             0,
976             0});
977             this.NumericGenMem.ValueChanged += new System.EventHandler(
                    this.NumericGenMem_ValueChanged);
978             //
979             // comboSpace
980             //
981             this.comboSpace.FormattingEnabled = true;
982             this.comboSpace.Items.AddRange(new object[] {
983             "Normal",
984             "Memory"});
985             this.comboSpace.Location = new System.Drawing.Point(534,
                    523);
986             this.comboSpace.Name = "comboSpace";
987             this.comboSpace.Size = new System.Drawing.Size(75, 21);
988             this.comboSpace.TabIndex = 30;
989             this.comboSpace.Text = "Normal";
990             //
991             // Form1
992             //
993             this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F)
                    ;
994             this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
                    ;
995             this.ClientSize = new System.Drawing.Size(1266, 710);
996             this.Controls.Add(this.comboSpace);
997             this.Controls.Add(this.groupBox12);
998             this.Controls.Add(this.groupBox11);
999             this.Controls.Add(this.groupBox10);
1000            this.Controls.Add(this.groupBox9);
1001            this.Controls.Add(this.groupBox8);
1002            this.Controls.Add(this.groupBox7);
1003            this.Controls.Add(this.BTNClear);
1004            this.Controls.Add(this.groupBox6);
1005            this.Controls.Add(this.groupBox5);
1006            this.Controls.Add(this.groupBox4);
```

```
1007            this.Controls.Add(this.BTNZoomM);
1008            this.Controls.Add(this.BTNZoomP);
1009            this.Controls.Add(this.groupBox3);
1010            this.Controls.Add(this.groupBox2);
1011            this.Controls.Add(this.groupBox1);
1012            this.Controls.Add(this.flowLayoutPanel1);
1013            this.Controls.Add(this.menuStrip1);
1014            this.MainMenuStrip = this.menuStrip1;
1015            this.Name = "Form1";
1016            this.Text = "  ";
1017            ((System.ComponentModel.ISupportInitialize)(this.
                    PBAutomataSimulator)).EndInit();
1018            ((System.ComponentModel.ISupportInitialize)(this.CHHistogram
                    )).EndInit();
1019            this.groupBox1.ResumeLayout(false);
1020            this.groupBox2.ResumeLayout(false);
1021            this.groupBox2.PerformLayout();
1022            ((System.ComponentModel.ISupportInitialize)(this.TBSpeed)).
                    EndInit();
1023            this.flowLayoutPanel1.ResumeLayout(false);
1024            this.flowLayoutPanel1.PerformLayout();
1025            this.groupBox3.ResumeLayout(false);
1026            this.groupBox3.PerformLayout();
1027            this.groupBox4.ResumeLayout(false);
1028            this.groupBox4.PerformLayout();
1029            this.groupBox5.ResumeLayout(false);
1030            this.groupBox5.PerformLayout();
1031            ((System.ComponentModel.ISupportInitialize)(this.numericOnes
                    )).EndInit();
1032            this.groupBox6.ResumeLayout(false);
1033            this.groupBox7.ResumeLayout(false);
1034            this.groupBox7.PerformLayout();
1035            ((System.ComponentModel.ISupportInitialize)(this.numericCols
                    )).EndInit();
1036            ((System.ComponentModel.ISupportInitialize)(this.numericRows
                    )).EndInit();
1037            this.groupBox8.ResumeLayout(false);
1038            this.groupBox9.ResumeLayout(false);
1039            this.groupBox9.PerformLayout();
1040            this.groupBox10.ResumeLayout(false);
1041            this.groupBox10.PerformLayout();
1042            ((System.ComponentModel.ISupportInitialize)(this.
                    pictureBox10)).EndInit();
1043            ((System.ComponentModel.ISupportInitialize)(this.pictureBox9
                    )).EndInit();
1044            ((System.ComponentModel.ISupportInitialize)(this.pictureBox8
                    )).EndInit();
1045            ((System.ComponentModel.ISupportInitialize)(this.pictureBox7
                    )).EndInit();
1046            ((System.ComponentModel.ISupportInitialize)(this.pictureBox6
                    )).EndInit();
1047            ((System.ComponentModel.ISupportInitialize)(this.pictureBox5
                    )).EndInit();
```

```
1048            ((System.ComponentModel.ISupportInitialize)(this.pictureBox4
                    )).EndInit();
1049            ((System.ComponentModel.ISupportInitialize)(this.pictureBox3
                    )).EndInit();
1050            ((System.ComponentModel.ISupportInitialize)(this.pictureBox2
                    )).EndInit();
1051            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1
                    )).EndInit();
1052            this.groupBox11.ResumeLayout(false);
1053            this.menuStrip1.ResumeLayout(false);
1054            this.menuStrip1.PerformLayout();
1055            this.groupBox12.ResumeLayout(false);
1056            this.groupBox12.PerformLayout();
1057            ((System.ComponentModel.ISupportInitialize)(this.
                    NumericGenMem)).EndInit();
1058            this.ResumeLayout(false);
1059            this.PerformLayout();
1060
1061        }
1062
1063        #endregion
1064
1065        private System.Windows.Forms.PictureBox PBAutomataSimulator;
1066        private System.Windows.Forms.DataVisualization.Charting.Chart
                CHHistogram;
1067        private System.Windows.Forms.GroupBox groupBox1;
1068        private System.Windows.Forms.Button BTNStart;
1069        private System.Windows.Forms.GroupBox groupBox2;
1070        private System.Windows.Forms.Label TXTGeneration;
1071        private System.Windows.Forms.Label label5;
1072        private System.Windows.Forms.Label label4;
1073        private System.Windows.Forms.Label label3;
1074        private System.Windows.Forms.Label label2;
1075        private System.Windows.Forms.ComboBox ComboBY2i;
1076        private System.Windows.Forms.ComboBox ComboBX2i;
1077        private System.Windows.Forms.ComboBox ComboBYi;
1078        private System.Windows.Forms.ComboBox ComboBXi;
1079        private System.Windows.Forms.Label TXTPopulation;
1080        private System.Windows.Forms.Button BTNStep;
1081        private System.Windows.Forms.TrackBar TBSpeed;
1082        private System.Windows.Forms.Timer TimerSimulation;
1083        private System.Windows.Forms.GroupBox groupBox3;
1084        private System.Windows.Forms.Label label6;
1085        private System.Windows.Forms.Label label1;
1086        private System.Windows.Forms.Button BTNZoomP;
1087        private System.Windows.Forms.Button BTNZoomM;
1088        private System.Windows.Forms.GroupBox groupBox4;
1089        private System.Windows.Forms.GroupBox groupBox5;
1090        private System.Windows.Forms.Button button1;
1091        private System.Windows.Forms.NumericUpDown numericOnes;
1092        private System.Windows.Forms.Label label7;
1093        private System.Windows.Forms.GroupBox groupBox6;
1094        private System.Windows.Forms.Button BTNSelectFile;
```

```
1095            private System.Windows.Forms.Button BTNClear;
1096            private System.Windows.Forms.GroupBox groupBox7;
1097            private System.Windows.Forms.NumericUpDown numericCols;
1098            private System.Windows.Forms.NumericUpDown numericRows;
1099            private System.Windows.Forms.Label label11;
1100            private System.Windows.Forms.Label label8;
1101            private System.Windows.Forms.Button BTNCreateMatrix;
1102            private System.Windows.Forms.GroupBox groupBox8;
1103            private System.Windows.Forms.Button BTNSave;
1104            private System.Windows.Forms.Label label12;
1105            private System.Windows.Forms.Label label13;
1106            private System.Windows.Forms.Label label14;
1107            private System.Windows.Forms.GroupBox groupBox9;
1108            private System.Windows.Forms.CheckBox CheckGraphEnabled;
1109            private System.Windows.Forms.GroupBox groupBox10;
1110            public System.Windows.Forms.FlowLayoutPanel flowLayoutPanel1;
1111            private System.Windows.Forms.PictureBox pictureBox1;
1112            private System.Windows.Forms.PictureBox pictureBox10;
1113            private System.Windows.Forms.PictureBox pictureBox9;
1114            private System.Windows.Forms.PictureBox pictureBox8;
1115            private System.Windows.Forms.PictureBox pictureBox7;
1116            private System.Windows.Forms.PictureBox pictureBox6;
1117            private System.Windows.Forms.PictureBox pictureBox5;
1118            private System.Windows.Forms.PictureBox pictureBox4;
1119            private System.Windows.Forms.PictureBox pictureBox3;
1120            private System.Windows.Forms.PictureBox pictureBox2;
1121            private System.Windows.Forms.GroupBox groupBox11;
1122            private System.Windows.Forms.ColorDialog colorDialog;
1123            private System.Windows.Forms.Button BTNGrid;
1124            private System.Windows.Forms.Button BTNDeadCells;
1125            private System.Windows.Forms.Button BTNAliveCells;
1126            private System.Windows.Forms.CheckBox CBPoints;
1127            private System.Windows.Forms.MenuStrip menuStrip1;
1128            private System.Windows.Forms.ToolStripMenuItem
                    generarPatronesToolStripMenuItem;
1129            private System.Windows.Forms.ToolStripMenuItem
                    generatePatternsToolStripMenuItem;
1130            private System.Windows.Forms.Button BTNSavePatterns;
1131            private System.Windows.Forms.CheckBox CBPatternRecognition;
1132            private System.Windows.Forms.GroupBox groupBox12;
1133            private System.Windows.Forms.Label label10;
1134            private System.Windows.Forms.Label label9;
1135            private System.Windows.Forms.ComboBox comboRuleMem;
1136            private System.Windows.Forms.NumericUpDown NumericGenMem;
1137            private System.ComponentModel.BackgroundWorker backgroundWorker1
                    ;
1138            private System.Windows.Forms.ComboBox comboSpace;
1139        }
1140    }
```

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          /*
                ****************************************************************

61           *                          Second space
62           *
                 ************************************************************
                 */
63
64          private uint[,] second_space;
65          private int TAO = 0;
66
67          /// <summary>
68          /// Constructor
69          /// </summary>
70          public Form1()
71          {
72              //Creating UI elements
73              InitializeComponent();
74              //We init all the predefined figures
75              initMosaics();
76              //Init the program with a 100, 100 matrix
77              createMatrix(5, 5);
78              //Make a responsive GUI
79              scrollBox();
80              //Creating each dictionary for our patterns
81              for (int i = 0; i < 6; i++) {
82                  patterns.Add(new Dictionary<ulong, ulong>());
83              }
84              //Getting tao value
85              TAO = (int)NumericGenMem.Value - 1;
86          }
87
88          private Color getColor() {
89              DialogResult result = colorDialog.ShowDialog();
90              return colorDialog.Color;
91          }
92          /// <summary>
```

```
93          /// This function creates a matrix of bools which size it's n x
                m
94          /// also this function adds an extra pair of cols and rows to
95          /// simulate a toroid
96          /// </summary>
97          /// <param name="rows"></param>
98          /// <param name="cols"></param>
99          private void createMatrix(int rows, int cols)
100         {
101             matrix = new uint[cols, rows];
102             //This matrix contains the second space
103             second_space = new uint[cols, rows];
104             scrollBox();
105         }
106
107         /// <summary>
108         /// This method paints the matrix in the Paint Box
109         /// </summary>
110         /// <param name="sender"></param>
111         /// <param name="e"></param>
112         private void PBAutomataSimulator_Paint(object sender,
                PaintEventArgs e)
113         {
114             Graphics graphics = e.Graphics;
115             if (Helper.getSpaceType(comboSpace.Text) == Helper.NORMAL)
116                 paintNormalSpace(graphics);
117             else
118             {
119                 if (TAO <= 0)
120                 {
121                     paintMemorySpace(graphics);
122                     TAO = (int)NumericGenMem.Value;
123                 }
124                 else
125                     paintNormalSpace(graphics);
126                 TAO--;
127             }
128         }
129
130         public void paintNormalSpace(Graphics graphics) {
131             int x_size = matrix.GetLength(0);
132             int y_size = matrix.GetLength(1);
133             total_cells = x_size * y_size;
134
135             for (int row = 0; row < x_size; row++)
136             {
137
138                 for (int col = 0; col < y_size; col++)
139                 {
140
141                     if (matrix[row, col] != DEAD)
142                     {
143                         SolidBrush aliveCellColor = new SolidBrush(
```

```
                        ColorHandler.fromIntToGradient(matrix[row,
                            col], ALIVE));
144                     graphics.FillRectangle(aliveCellColor, row *
                            cellArea, col * cellArea, cellArea, cellArea
                            );
145                 }
146                 else
147                     graphics.FillRectangle(dead, row * cellArea, col
                            * cellArea, cellArea, cellArea);
148             }
149         }
150
151         for (int y = 0; y < y_size; y++)
152         {
153             graphics.DrawLine(grid, 0, y * cellArea, total_cells *
                    cellArea, y * cellArea);
154         }
155
156         for (int x = 0; x < x_size; x++)
157         {
158             graphics.DrawLine(grid, x * cellArea, 0, x * cellArea,
                    total_cells * cellArea);
159         }
160     }
161
162     public void paintMemorySpace(Graphics graphics) {
163         int x_size = second_space.GetLength(0);
164         int y_size = second_space.GetLength(1);
165         total_cells = x_size * y_size;
166
167         for (int row = 0; row < x_size; row++)
168         {
169             for (int col = 0; col < y_size; col++)
170             {
171                 uint current_cell = second_space[row, col];
172                 switch (Helper.getRule(comboRuleMem.Text)) {
173                     case Helper.MAJORITY:
174                         current_cell = Helper.getMajority(
                                second_space[row, col], (int)(
                                NumericGenMem.Value)) ;
175                         drawCell(graphics, current_cell, row, col);
176                     break;
177
178                     case Helper.MINORITY:
179                         current_cell = Helper.getMinority(
                                second_space[row, col], (int)(
                                NumericGenMem.Value));
180                         drawCell(graphics, current_cell, row, col);
181                         break;
182
183                     default:
184                         current_cell = Helper.getParity(second_space
                                [row, col], (int)(NumericGenMem.Value));
```

```
185                         drawCell(graphics, current_cell, row, col);
186                         break;
187                     }
188                     matrix[row, col] = (current_cell == 1) ? ALIVE :
                            DEAD;
189                     second_space[row, col] = 0;
190                 }
191             }
192             for (int y = 0; y < y_size; y++)
193             {
194                 graphics.DrawLine(grid, 0, y * cellArea, total_cells *
                        cellArea, y * cellArea);
195             }
196
197             for (int x = 0; x < x_size; x++)
198             {
199                 graphics.DrawLine(grid, x * cellArea, 0, x * cellArea,
                        total_cells * cellArea);
200             }
201         }
202
203         public void drawCell(Graphics graphics, uint state, int row, int
                col) {
204             if (state == 1)
205                 graphics.FillRectangle(alive, row * cellArea, col *
                        cellArea, cellArea, cellArea);
206             else
207                 graphics.FillRectangle(dead, row * cellArea, col *
                        cellArea, cellArea, cellArea);
208         }
209         /// <summary>
210         /// This funcion manipulates a matrix and evaluate it
211         /// using our rules.
212         /// </summary>
213         /// <param name="p_matrix"></param>
214         /// <returns>A matrix with the new generation data</returns>
215         private uint[,] nextGeneration(uint[,] p_matrix)
216         {
217
218             /*****************************************************
219              *                    CONDITIONS                     *
220              * ***************************************************/
221             //23 33 GAME OF LIFE
222             //77 22 DIFFUSION
223             /****************** X values *******************/
224             int Xi = Int32.Parse(string.IsNullOrEmpty(ComboBXi.Text) ? "
                    2" : ComboBXi.Text);
225
226             /****************** Y values *******************/
227             int Yi = Int32.Parse(string.IsNullOrEmpty(ComboBYi.Text) ? "
                    3" : ComboBYi.Text);
228
229             /****************** X2 values ******************/
```

```
230             int X2i = Int32.Parse((string.IsNullOrEmpty(ComboBX2i.Text)
                    ? "3" : ComboBX2i.Text));
231
232             /****************** Y2 values ******************/
233             int Y2i = Int32.Parse((string.IsNullOrEmpty(ComboBY2i.Text)
                    ? "3" : ComboBY2i.Text));
234
235             uint[,] new_matrix = new uint[p_matrix.GetLength(0),
                    p_matrix.GetLength(1)];
236             //We check each cell from the original matrix and we
                    substitute it
237             for (int row = 0; row < p_matrix.GetLength(0); row++)
238             {
239
240                 for (int col = 0; col < p_matrix.GetLength(1); col++)
241                 {
242                     /**
243                      * Here we need to evaluate using the rules given by
                             input
244                      **/
245                     int neighbors = getAliveNeighbors(p_matrix, row, col
                            );
246                     uint color_cell = p_matrix[row, col] - 1;
247                     //If the cell is alive
248                     if (p_matrix[row, col] != DEAD)
249                     {
250
251                         new_matrix[row, col] = (neighbors >= Xi &&
                                neighbors <= Yi) ? (color_cell) : DEAD;
252                     }
253                     //If the central cell is dead
254                     else
255                     {
256                         new_matrix[row, col] = (neighbors >= X2i &&
                                neighbors <= Y2i) ? ALIVE : DEAD;
257                     }
258                     //Counting the ones in our second space
259                     if ((new_matrix[row, col] != DEAD) && Helper.
                            getSpaceType(comboSpace.Text) == Helper.MEMORY)
                            {
260                         second_space[row, col]++;
261                     }
262                 }
263
264             }
265
266             return new_matrix;
267         }
268
269         /// <summary>
270         /// Gets information about the cells around a central cell.
271         /// Obviouslly the cells must to be alive.
272         /// </summary>
```

```
273         /// <param name="p_matrix">The actual matrix</param>
274         /// <param name="p_row">row of the central cell</param>
275         /// <param name="p_col">col of the central cell</param>
276         /// <returns>Number of neighboors around the central cell (just
                living neighbors)</returns>
277         private int getAliveNeighbors(uint[,] p_matrix, int p_row, int
                p_col)
278         {
279
280             int neighbors = 0;
281             int max_x = p_matrix.GetLength(0);
282             int max_y = p_matrix.GetLength(1);
283             uint[,] sub_matrix = new uint[3, 3];
284
285             for (int row = -1, sx = 0; row <= 1; row++, sx++)
286             {
287
288                 for (int col = -1, sy = 0; col <= 1; col++, sy++)
289                 {
290
291                     int x = row + p_row;
292                     int y = col + p_col;
293
294                     //We are in the center cell
295                     if (x == p_row && y == p_col)
296                     {
297                         sub_matrix[sx, sy] = p_matrix[x, y];
298                         continue;
299                     }
300                     // ----- Corners ----- //
301
302                     //Up-Left
303                     if (x == -1 && y == -1 && (p_matrix[max_x - 1, max_y
                            - 1] != DEAD))
304                     {
305                         sub_matrix[sx, sy] = p_matrix[max_x - 1, max_y -
                                1];
306                         neighbors++;
307                     }
308                     //Down-Right
309                     if (x == max_x && y == max_y && (p_matrix[0, 0] !=
                            DEAD))
310                     {
311                         sub_matrix[sx, sy] = p_matrix[0, 0];
312                         neighbors++;
313                     }
314                     //Up-Right
315                     if (x == -1 && y == max_y && (p_matrix[max_x - 1, 0]
                            != DEAD))
316                     {
317                         sub_matrix[sx, sy] = p_matrix[max_x - 1, 0];
318                         neighbors++;
319                     }
```

```
320                     //Down-left
321                     if (x == max_x && y == -1 && (p_matrix[0, max_y - 1]
                            != DEAD))
322                     {
323                         sub_matrix[sx, sy] = p_matrix[0, max_y - 1];
324                         neighbors++;
325                     }
326                     // ------ Edges ------ //
327
328                     if (y >= 0 && y < max_y)
329                     {
330                         //Up
331                         if (x == -1 && p_matrix[max_x - 1, y] != DEAD)
332                         {
333                             sub_matrix[sx, sy] = p_matrix[max_x - 1, y];
334                             neighbors++;
335                         }
336                         //Down
337                         else if (x == max_x && p_matrix[0, y] != DEAD)
338                         {
339                             sub_matrix[sx, sy] = p_matrix[0, y];
340                             neighbors++;
341                         }
342                     }
343                     if (x >= 0 && x < max_x)
344                     {
345                         //Right
346                         if (y == -1 && p_matrix[x, max_y - 1] != DEAD)
347                         {
348                             sub_matrix[sx, sy] = p_matrix[x, max_y - 1];
349                             neighbors++;
350                         }
351                         //Left
352                         else if (y == max_y && p_matrix[x, 0] != DEAD)
353                         {
354                             sub_matrix[sx, sy] = p_matrix[x, 0];
355                             neighbors++;
356                         }
357                     }
358
359                     if (x < 0 || x >= max_x)
360                     {
361                         continue;
362                     }
363
364                     if (y < 0 || y >= max_y)
365                     {
366                         continue;
367                     }
368
369                     if (p_matrix[x, y] != DEAD)
370                     {
371                         sub_matrix[sx, sy] = p_matrix[x, y];
```

```
372                         neighbors++;
373                     }
374
375                 }
376             }
377             if (CBPatternRecognition.Checked) {
378                 patternRecognition(generateMatrixPatterns(p_matrix,
                        p_row, p_col, 2));
379                 patternRecognition(sub_matrix);
380                 patternRecognition(generateMatrixPatterns(p_matrix,
                        p_row, p_col, 4));
381             }
382             return neighbors;
383         }
384
385         /// <summary>
386         /// This method calls nextGeneration method and
387         /// updates the GUI and the count of our alive cells
388         /// </summary>
389         private void step()
390         {
391             matrix = nextGeneration(matrix);
392             updateTextGeneration();
393             countOnes();
394             if (Helper.getSpaceType(comboSpace.Text) == Helper.NORMAL)
395             {
396                 PBAutomataSimulator.Invalidate();
397             }
398             else
399             {
400                 PBAutomataSimulator.Invalidate();
401             }
402
403         }
404
405         /// <summary>
406         /// Here we just change the text that show us
407         /// the number of generations
408         /// </summary>
409         private void updateTextGeneration()
410         {
411             TXTGeneration.Text = "Generation: " + generation++;
412         }
413
414         /// <summary>
415         /// This method make a rezise of the Paint Box and flow layout
                panel
416         /// it makes possible make zoom and the movement into the GUI
417         /// </summary>
418         private void scrollBox()
419         {
420             PBAutomataSimulator.Size = new Size((matrix.GetLength(0)) *
                    cellArea, (matrix.GetLength(1)) * cellArea);
```

```
421             PBAutomataSimulator.SizeMode = PictureBoxSizeMode.AutoSize;
422             flowLayoutPanel1.AutoScroll = true;
423             flowLayoutPanel1.Controls.Add(PBAutomataSimulator);
424         }
425 
426         /// <summary>
427         /// As you can imagine here we just get the number of ones
428         /// in our matrix (alive cells)
429         /// </summary>
430         private void countOnes()
431         {
432             int ones = 0;
433             for (int x = 0; x < matrix.GetLength(0); x++)
434             {
435 
436                 for (int y = 0; y < matrix.GetLength(1); y++)
437                 {
438                     if (matrix[x, y] != DEAD) ones++;
439                 }
440             }
441 
442             if (CheckGraphEnabled.Checked)
443                 CHHistogram.Series["#Ones"].Points.AddY(ones);
444             TXTPopulation.Text = "Population " + ones;
445             acumOnes += ones;
446             double val = acumOnes / generation;
447             label12.Text = "Total Population: " + acumOnes;
448             label13.Text = "Average: " + (val);
449             label14.Text = "Density: " + (val / (matrix.GetLength(0) *
                    matrix.GetLength(1)));
450         }
451 
452         /*
                ************************************************************************
453          *                                      SECOND TERM CODE                    *
454          ************************************************************************
                 */
455 
456         /// <summary>
457         /// This funciton generates multiple binary string one per each
458         /// matrix dimension. For our case the maximum matrix will be
459         /// of 7 x 7 at most
460         /// </summary>
461         private void generatePatterns(int size)
462         {
463             data.Clear();
464             recurrences.Clear();
465             clasifications.Clear();
466             Console.WriteLine("generatePatterns(" + size + ")");
467             //To generate all the possible combinations inside
468             //a matrix from 2x2 to 8x8 (just square) we gonna
```

```
469             //to convert from a decimal number to a binary string
470             //so, size_string contains the limit of combinations
471             //in each matrix n^2 where n is the size of each matrix
472             try
473             {
474                 int size_string = 0;
475                 ulong n_combinations = 0;
476                 SaveFileDialog saveFileDialog = new SaveFileDialog();
477                 saveFileDialog.Filter = "Archivo de texto|*.txt";
478                 saveFileDialog.Title = "Patterns file name";
479                 saveFileDialog.ShowDialog();
480                 StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
481                 //Size of the binary string: 2^2, 3^2, 4^2, ... , 7^2
482                 size_string = size * size;
483                 //Now we get the number of combinations it is 2^
                        size_string
484                 n_combinations = (ulong)Math.Pow(2.0, size_string);
485                 sw.Write("GraphPlot[{");
486                 //We iterate from 0 to 2^n and convert this number to a
                        binary string
487                 for (ulong j = 1; j < n_combinations; j++)
488                 {
489                     //We convert j to a binary string
490                     string str_binary = Convert.ToString((long)j, 2);
491                     while (str_binary.Length != size_string)
492                     {
493                         str_binary = "0" + str_binary;
494                     }
495                     uint[,] next_state = nextGeneration(
                            fromBinaryToMatrix(str_binary));
496                     string str_next_state = fromMatrixToString(
                            next_state);
497                     ulong nextState = Convert.ToUInt64(str_next_state,
                            2);
498                     if (saveFileDialog != null) {
499                         sw.Write(j + "->" + nextState + ((j <
                                n_combinations - 1) ? ", " : ""));
500                         data.Add(j, nextState);
501                     }
502                 }
503                 sw.Write("}]");
504                 sw.Close();
505                 MessageBox.Show("I've stored something");
506                 //Sorting the paterns created
507                 sortTransitions();
508                 //Create some objects with the paterns
509                 CreateGraphObjects();
510                 //We create a file to send it to mathematica
511                 outputMathematica();
512             }
513             catch (Exception e) {
514                 Console.WriteLine("An exception has occurred on
```

```
                generatePatterns " + e);
515         }
516     }
517
518     /// <summary>
519     /// Generate a txt file with the structure of a mathematica file
520     /// </summary>
521     private void outputMathematica() {
522         Dictionary<ulong, ulong> unique_nodes = new Dictionary<ulong
                , ulong>();
523         try {
524             SaveFileDialog saveFileDialog = new SaveFileDialog();
525             saveFileDialog.Filter = "Archivo de texto|*.txt";
526             saveFileDialog.Title = "Paterns filtered";
527             saveFileDialog.ShowDialog();
528             StreamWriter sw = new StreamWriter(saveFileDialog.
                    OpenFile());
529
530             sw.Write("GraphPlot[{");
531             foreach (KeyValuePair<string, Graph> item_graph in
                    clasifications) {
532                 Graph current_graph = item_graph.Value;
533                 Dictionary<ulong, List<ulong>> node = current_graph.
                        getAllNodes();
534                 foreach (KeyValuePair<ulong, List<ulong>> item in
                        node) {
535
536                     for (int i = 0; i < item.Value.Count; i++) {
537                         if (!unique_nodes.ContainsKey(item.Value[i])
                                )
538                             unique_nodes.Add(item.Value[i], item.Key
                                    );
539                     }
540                 }
541             }
542
543             foreach (KeyValuePair<ulong, ulong> item in unique_nodes
                    )
544                 sw.Write(item.Key + "->" + item.Value + ((item.Key
                        == unique_nodes.Last().Key) ? "" : ","));
545             sw.Write("}]");
546             sw.Close();
547             MessageBox.Show("I've stored something  ");
548         }
549         catch (Exception e) {
550             Console.WriteLine("An exception has occurred creating a
                    mathematica file " + e);
551         }
552     }
553     /// <summary>
554     /// This function generate a matrix using a binary string.
555     /// We are just considering a square matrix, so we can
556     /// calculate the number of rows and cols calculating
```

```
557         /// the square root of the binary_string size
558         /// </summary>
559         /// <param name="binary_string">This string
560         /// contains a matrix but in a dimension</param>
561         private uint[,] fromBinaryToMatrix(string binary_string)
562         {
563             //We get the size of our sub_matrix. As we know
564             //the matrix it's a square, so we need to calculate
565             //the square root of the length of the binary string
566             int size = Convert.ToInt16(Math.Sqrt(binary_string.Length));
567             //We create a new boolean matrix
568             uint[,] sub_matrix = new uint[size, size];
569             //We build the sub matrix using our binary string
570             for (int x = 0, position = 0; x < size; x++)
571             {
572                 for (int y = 0; y < size; y++)
573                 {
574                     //We add an element int the x'th row in the y'th
                            position
575                     //If this element it's equals to an one we put true
                            in out sub matrix
576                     sub_matrix[x, y] = (binary_string[position++] == '0'
                            ) ? DEAD : ALIVE;
577 
578                 }
579 
580             }
581             return sub_matrix;
582         }
583 
584         /// <summary>
585         /// Convert a matrix to a binary string
586         /// </summary>
587         /// <param name="a_matrix">Source matrix</param>
588         /// <returns>Binary string</returns>
589         private string fromMatrixToString(uint[,] a_matrix)
590         {
591             string str = "";
592 
593             for (int i = 0; i < a_matrix.GetLength(0); i++) {
594                 for (int j = 0; j < a_matrix.GetLength(1); j++) {
595                     str += (a_matrix[i, j] == DEAD ? "0" : "1");
596                 }
597             }
598 
599             return str;
600         }
601 
602         /// <summary>
603         /// Just for testing
604         /// </summary>
605         /// <param name="a_matrix">Source matrix</param>
606         private void printMatrix(int[,] a_matrix) {
```

```
607
608             Console.WriteLine("
                    ------------------------------------------------------ ");
609             for (int i = 0; i < a_matrix.GetLength(0); i++) {
610
611                 for (int j = 0; j < a_matrix.GetLength(1); j++) {
612                     Console.Write(" " + ((a_matrix[i, j] == DEAD) ? "1"
                            : "0"));
613                 }
614                 Console.WriteLine();
615             }
616             Console.WriteLine("
                    ------------------------------------------------------ ");
617         }
618         /// <summary>
619         /// This function
620         /// </summary>
621         /// <param name="sub_matrix"></param>
622         private void patternRecognition(uint[,] p_matrix)
623         {
624             int dimension = p_matrix.GetLength(0);
625             ulong key = Convert.ToUInt64(fromMatrixToString(p_matrix),
                    2);
626
627             if (!patterns[dimension].ContainsKey(key))
628                 patterns[dimension].Add(key, 1);
629             else
630                 patterns[dimension][key]++;
631         }
632
633         private uint[,] generateMatrixPatterns(uint[,] p_matrix, int
                p_row, int p_col, int dimension) {
634             int init = -1;
635             int end = 0;
636
637             if (dimension == 4)
638             {
639                 init = -2;
640                 end = 1;
641             }
642
643             int max_x = p_matrix.GetLength(0);
644             int max_y = p_matrix.GetLength(1);
645             uint[,] sub_matrix = new uint[dimension, dimension];
646
647             for (int row = init, sx = 0; row <= end; row++, sx++)
648             {
649
650                 for (int col = init, sy = 0; col <= end; col++, sy++)
651                 {
652
653                     int x = row + p_row;
654                     int y = col + p_col;
```

```
655
656                         //We are in the center cell
657                         if (x == p_row && y == p_col)
658                         {
659                             sub_matrix[sx, sy] = p_matrix[x, y];
660                             continue;
661                         }
662                         // ----- Corners ----- //
663
664                         //Up-Left
665                         if (x < 0 && y < 0 && (p_matrix[max_x + x, max_y + y
                                ] != DEAD))
666                         {
667                             sub_matrix[sx, sy] = p_matrix[max_x + x, max_y +
                                    y];
668                         }
669                         //Down-Right
670                         if (x >= max_x && y >= max_y && (p_matrix[x - max_x,
                                 y - max_y] != DEAD))
671                         {
672                             sub_matrix[sx, sy] = p_matrix[x - max_x, y -
                                    max_y];
673                         }
674                         //Up-Right
675                         if (x < 0 && y == max_y && (p_matrix[max_x + x, 0]
                                != DEAD))
676                         {
677                             sub_matrix[sx, sy] = p_matrix[max_x + x, 0];
678                         }
679                         //Down-left
680                         if (x >= max_x && y < 0 && (p_matrix[x - max_x,
                                max_y + y] != DEAD))
681                         {
682                             sub_matrix[sx, sy] = p_matrix[x - max_x, max_y +
                                    y];
683                         }
684                         // ----- Edges ----- //
685
686                         if (y >= 0 && y < max_y)
687                         {
688                             //Up
689                             if (x < 0 && p_matrix[x + max_x, y] != DEAD)
690                             {
691                                 sub_matrix[sx, sy] = p_matrix[x + max_x, y];
692                             }
693                             //Down
694                             else if (x >= max_x && p_matrix[x - max_x, y] !=
                                    DEAD)
695                             {
696                                 sub_matrix[sx, sy] = p_matrix[x - max_x, y];
697                             }
698                         }
699                         if (x >= 0 && x < max_x)
```

```
700                     {
701                         //Right
702                         if (y < 0 && p_matrix[x, y + max_y] != DEAD)
703                         {
704                             sub_matrix[sx, sy] = p_matrix[x, y + max_y];
705                         }
706                         //Left
707                         else if (y >= max_y && p_matrix[x, y - max_y] !=
                                DEAD)
708                         {
709                             sub_matrix[sx, sy] = p_matrix[x, y - max_y];
710                         }
711                     }
712 
713                     if (x < 0 || x >= max_x)
714                     {
715                         continue;
716                     }
717 
718                     if (y < 0 || y >= max_y)
719                     {
720                         continue;
721                     }
722 
723                     if (p_matrix[x, y] != DEAD)
724                     {
725                         sub_matrix[sx, sy] = p_matrix[x, y];
726                     }
727 
728                 }
729             }
730             return sub_matrix;
731         }
732 
733         private void storePatterns(int dimension) {
734             try {
735                 SaveFileDialog saveFileDialog = new SaveFileDialog();
736                 saveFileDialog.Filter = "Archivo de texto|*.txt";
737                 saveFileDialog.Title = "Patterns found " + dimension + "
                        x" + dimension;
738                 saveFileDialog.ShowDialog();
739                 StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
740                 sw.WriteLine("---------- Patterns found in a " +
                        dimension + " x " + dimension + " matrix ----------"
                        );
741                 foreach (KeyValuePair<ulong, ulong> item in patterns[
                        dimension]) {
742                     sw.WriteLine(item.Key + " appears " + item.Value + "
                            times ");
743                 }
744                 sw.Close();
745             }
```

```
746             catch (Exception e) {
747                 Console.WriteLine("An exception has occurred on
                        storePatterns() " + e);
748             }
749         }
750 
751         /// <summary>
752         /// This function intialize our predefined patterns. This
                patterns
753         /// allow us to draw and drop patterns into the automata space.
754         /// </summary>
755         private void initMosaics()
756         {
757 
758             figure.Add(null);
759 
760             figure.Add(new uint[,] { { ALIVE, ALIVE},
761                                      { ALIVE, ALIVE } });
762 
763             figure.Add(new uint[,] { { ALIVE, ALIVE, DEAD, ALIVE, ALIVE
                    },
764                                      { ALIVE, DEAD, DEAD, DEAD, ALIVE },
765                                      { DEAD, ALIVE, ALIVE, ALIVE, DEAD }
                                            });
766 
767             figure.Add(new uint[,] { { ALIVE, ALIVE, DEAD },
768                                      { ALIVE, DEAD, ALIVE },
769                                      { DEAD, ALIVE, ALIVE } });
770 
771             figure.Add(new uint[,] { { DEAD, ALIVE, DEAD },
772                                      { DEAD, DEAD, ALIVE },
773                                      { ALIVE, ALIVE, ALIVE } });
774 
775             figure.Add(new uint[,] { { DEAD, ALIVE, DEAD },
776                                      { ALIVE, DEAD, ALIVE},
777                                      { DEAD, ALIVE, DEAD} });
778 
779             figure.Add(new uint[,] { { DEAD, ALIVE, ALIVE, ALIVE, ALIVE,
                    ALIVE, ALIVE },
780                                      { ALIVE, DEAD, DEAD, DEAD, DEAD,
                                            DEAD, ALIVE },
781                                      { DEAD, DEAD, DEAD, DEAD, DEAD,
                                            DEAD, ALIVE},
782                                      { ALIVE, DEAD, DEAD, DEAD, DEAD,
                                            ALIVE, DEAD },
783                                      { DEAD, DEAD, ALIVE, ALIVE, DEAD,
                                            DEAD, DEAD } });
784 
785             figure.Add(new uint[,] { { ALIVE, DEAD, DEAD },
786                                      { DEAD, DEAD, ALIVE },
787                                      { DEAD, DEAD, ALIVE },
788                                      { ALIVE, DEAD, DEAD } });
789 
```

```
790             figure.Add(new uint[,] { { DEAD, ALIVE, ALIVE, DEAD },
791                                      { ALIVE, DEAD, DEAD, DEAD },
792                                      { ALIVE, DEAD, DEAD, DEAD },
793                                      { DEAD, ALIVE, ALIVE, DEAD } });
794
795             figure.Add(new uint[,] { { ALIVE, ALIVE, ALIVE } });
796
797             figure.Add(new uint[,] { { DEAD, DEAD, ALIVE },
798                                      { ALIVE, DEAD, DEAD },
799                                      { ALIVE, DEAD, DEAD },
800                                      { DEAD, ALIVE, DEAD } });
801
802         }
803
804         /// <summary>
805         /// Sort the elements of out Dictionary called data.
806         /// The elements are sorted using the Value
807         /// </summary>
808         private void sortTransitions()
809         {
810
811             foreach (var item in data.OrderByDescending(key => key.Value
                    ))
812             {
813                 if (!recurrences.ContainsKey(item.Value))
814                 {
815                     List<ulong> aux = new List<ulong>();
816                     aux.Add(item.Key);
817                     recurrences.Add(item.Value, aux);
818                 }
819                 else
820                     recurrences[item.Value].Add(item.Key);
821             }
822
823         }
824
825         private void CreateGraphObjects()
826         {
827             //Itering into each element of the Dictionary
828             foreach (KeyValuePair<ulong, List<ulong>> item in
                    recurrences)
829             {
830                 Dictionary<ulong, List<ulong>> aux = new Dictionary<
                        ulong, List<ulong>>();
831                 aux.Add(item.Key, item.Value);
832                 //Itering through each element of the list
833                 for (int i = 0; i < item.Value.Count; i++)
834                 {
835                     ulong element_list = item.Value[i];
836                     if (recurrences.ContainsKey(element_list))
837                     {
838                         if (!aux.ContainsKey(element_list))
839                             aux.Add(element_list, recurrences[
```

```
                                element_list]);
840                         }
841
842                     }
843                     Graph graph_element = new Graph(aux);
844                     //List<ulong> key = graph_element.getKey();
845                     string key = graph_element.getKey();
846                     if (aux.Count > 0 && !clasifications.ContainsKey(key))
847                         clasifications.Add(key, graph_element);
848                 }
849             }
850
851             /// <summary>
852             /// Comer if two list are equal
853             /// </summary>
854             /// <param name="first_list">First list</param>
855             /// <param name="second_list">Second list</param>
856             /// <returns>true if the lists are equals</returns>
857             private bool compareTwoList(List<ulong> first_list, List<ulong>
                    second_list)
858             {
859                 if (first_list.Count != second_list.Count)
860                     return false;
861
862                 for (int i = 0; i < first_list.Count; i++)
863                 {
864                     if (first_list[i] != second_list[i])
865                         return false;
866                 }
867                 return true;
868             }
869
870             /// <summary>
871             /// Just for testing
872             /// </summary>
873             private void printRecurrences()
874             {
875                 foreach (KeyValuePair<ulong, List<ulong>> item in
                        recurrences)
876                 {
877                     Console.WriteLine("Element " + item.Key);
878                     for (int i = 0; i < item.Value.Count; i++)
879                     {
880                         Console.Write(item.Value[i] + " ");
881                     }
882                     Console.WriteLine();
883                 }
884             }
885
886
887             /**************************************************
888              *                         Events                 *
889              **************************************************/
```

```
890
891         private void BTNStep_Click(object sender, EventArgs e)
892         {
893             step();
894         }
895
896         private void PBAutomataSimulator_MouseDown(object sender,
                MouseEventArgs e)
897         {
898             int x = e.X / cellArea;
899             int y = e.Y / cellArea;
900             matrix[x, y] = (matrix[x, y] == ALIVE) ? DEAD : ALIVE;
901             PBAutomataSimulator.Invalidate();
902         }
903
904         private void BTNStart_Click(object sender, EventArgs e)
905         {
906             if (BTNStart.Text == "Start")
907             {
908                 TimerSimulation.Start();
909                 BTNStart.Text = "Stop";
910             }
911             else
912             {
913                 TimerSimulation.Stop();
914                 BTNStart.Text = "Start";
915             }
916         }
917
918         private void trackBar1_ValueChanged(object sender, EventArgs e)
919         {
920             TimerSimulation.Interval = TBSpeed.Value;
921         }
922
923         private void TimerSimulation_Tick(object sender, EventArgs e)
924         {
925             step();
926         }
927
928         private void BTNZoomP_Click(object sender, EventArgs e)
929         {
930             if (cellArea < 50)
931             {
932                 cellArea++;
933                 PBAutomataSimulator.Invalidate();
934                 scrollBox();
935             }
936         }
937
938         private void BTNZoomM_Click(object sender, EventArgs e)
939         {
940             if (cellArea > 1)
941             {
```

```
942                 cellArea--;
943                 PBAutomataSimulator.Invalidate();
944                 scrollBox();
945             }
946         }
947 
948         private void button1_Click(object sender, EventArgs e)
949         {
950             Random r = new Random();
951             for (int x = 0; x < matrix.GetLength(0); x++)
952             {
953 
954                 for (int y = 0; y < matrix.GetLength(1); y++)
955                 {
956 
957                     float rand = r.Next(0, 100);
958                     if (rand < double.Parse(numericOnes.Text))
959                     {
960                         matrix[x, y] = ALIVE;
961                     }
962                     else matrix[x, y] = DEAD;
963                 }
964             }
965             PBAutomataSimulator.Invalidate();
966         }
967 
968         private void BTNClear_Click(object sender, EventArgs e)
969         {
970             CHHistogram.Series["#Ones"].Points.Clear();
971             for (int x = 0; x < matrix.GetLength(0); x++)
972             {
973 
974                 for (int y = 0; y < matrix.GetLength(1); y++)
975                 {
976                     matrix[x, y] = DEAD;
977                     second_space[x, y] = DEAD;
978                 }
979             }
980             TAO = (int)NumericGenMem.Value;
981             PBAutomataSimulator.Invalidate();
982 
983             acumOnes = generation = 0;
984             for (int i = 0; i < patterns.Count; i++) {
985                 patterns[i].Clear();
986             }
987         }
988 
989         private void BTNCreateMatrix_Click(object sender, EventArgs e)
990         {
991             int rows = (numericRows.Value == 0) ? 100 : (int)numericRows
                    .Value;
992             int cols = (numericCols.Value == 0) ? 100 : (int)numericCols
                    .Value;
```

```
993             createMatrix(rows, cols);
994         }
995
996         private void button2_Click(object sender, EventArgs e)
997         {
998             int min_lines = 0;
999             int min_chara = 0;
1000            String fileName = null;
1001
1002            try
1003            {
1004                using (OpenFileDialog openFileDialog = new
                        OpenFileDialog())
1005                {
1006                    openFileDialog.InitialDirectory = "c\\";
1007                    openFileDialog.Filter = "txt files (*.txt)|*.txt";
1008                    openFileDialog.FilterIndex = 2;
1009                    if (openFileDialog.ShowDialog() == DialogResult.OK)
1010                    {
1011                        fileName = openFileDialog.FileName;
1012                    }
1013                }
1014
1015                if (fileName != null)
1016                {
1017                    Console.WriteLine(fileName);
1018                    StreamReader objectReader = new StreamReader(
                            fileName);
1019                    //Reading the file, line per line
1020                    String line = "";
1021                    ArrayList arrayText = new ArrayList();
1022                    while (line != null)
1023                    {
1024                        line = objectReader.ReadLine();
1025                        if (line != null)
1026                            arrayText.Add(line);
1027                    }
1028                    objectReader.Close();
1029                    //Iterate into the ArrayList and send the
                            information to the GUI
1030                    min_lines = arrayText.Count;
1031                    min_chara = arrayText[0].ToString().Length;
1032
1033                    Console.WriteLine(min_chara);
1034                    Console.WriteLine(min_lines);
1035                    if (min_chara > matrix.GetLength(1) && min_lines >
                            matrix.GetLength(0))
1036                    {
1037                        createMatrix(min_chara, min_lines);
1038                    }
1039                    for (int i = 0; i < min_lines; i++)
1040                    {
1041                        string strlne = arrayText[i].ToString().Trim();
```

```
1042                        int j = 0;
1043
1044                        foreach (char c in strlne)
1045                        {
1046                            matrix[j++, i] = (c == '1')?ALIVE:DEAD;
1047                        }
1048                    }
1049                    Console.ReadLine();
1050                }
1051            }
1052            catch (Exception ex)
1053            {
1054                Console.WriteLine(ex);
1055            }
1056            PBAutomataSimulator.Invalidate();
1057        }
1058
1059        private void BTNSave_Click(object sender, EventArgs e)
1060        {
1061
1062            try
1063            {
1064                SaveFileDialog saveFileDialog = new SaveFileDialog();
1065                saveFileDialog.Filter = "Archivo de texto|*.txt";
1066                saveFileDialog.Title = "Actual state cellular automata";
1067                saveFileDialog.ShowDialog();
1068                if (saveFileDialog != null)
1069                {
1070                    StreamWriter sw = new StreamWriter(saveFileDialog.
                        OpenFile());
1071                    for (int i = 0; i < matrix.GetLength(0); i++)
1072                    {
1073
1074                        for (int j = 0; j < matrix.GetLength(1); j++)
1075                            sw.Write(((matrix[j, i] == DEAD) ? "1" : "0"
                                ));
1076                        sw.WriteLine();
1077                    }
1078                    sw.Close();
1079                    MessageBox.Show("I've stored something");
1080                }
1081            }
1082            catch (Exception ex)
1083            {
1084                Console.WriteLine(ex);
1085            }
1086
1087        }
1088
1089        private void PBAutomataSimulator_MouseMove(object sender,
            MouseEventArgs e)
1090        {
1091            try
```

```
1092            {
1093                if (move && index_pattern != 0)
1094                {
1095                    int x = e.X / cellArea;
1096                    int y = e.Y / cellArea;
1097
1098                    uint[,] draw_figure = figure[index_pattern];
1099                    int x_size = (draw_figure).GetLength(1);
1100                    int y_size = (draw_figure).GetLength(0);
1101                    initMosaics();
1102                    for (int x_c = 0; x_c < x_size; x_c++)
1103                    {
1104
1105                        for (int y_c = 0; y_c < y_size; y_c++)
1106                        {
1107                            matrix[x_c + x, y_c + y] = draw_figure[y_c,
                                x_c];
1108                        }
1109                    }
1110                    if (Helper.getSpaceType(comboSpace.Text) == Helper.
                        MEMORY)
1111                        TAO++;
1112                    PBAutomataSimulator.Invalidate();
1113                    move = false;
1114                    index_pattern = 0;
1115                }
1116            }
1117            catch (Exception) {
1118
1119            }
1120        }
1121
1122        private void pictureBox1_MouseUp(object sender, MouseEventArgs e
            )
1123        {
1124            move = true;
1125            index_pattern = 1;
1126        }
1127
1128        private void pictureBox2_MouseUp(object sender, MouseEventArgs e
            )
1129        {
1130            move = true;
1131            index_pattern = 2;
1132        }
1133
1134        private void pictureBox3_MouseUp(object sender, MouseEventArgs e
            )
1135        {
1136            move = true;
1137            index_pattern = 3;
1138        }
1139
```

```
1140        private void pictureBox4_MouseUp(object sender, MouseEventArgs e
                )
1141        {
1142            move = true;
1143            index_pattern = 4;
1144        }
1145
1146        private void pictureBox5_MouseUp(object sender, MouseEventArgs e
                )
1147        {
1148            move = true;
1149            index_pattern = 5;
1150        }
1151
1152        private void pictureBox6_MouseUp(object sender, MouseEventArgs e
                )
1153        {
1154            move = true;
1155            index_pattern = 6;
1156        }
1157
1158        private void pictureBox7_MouseUp(object sender, MouseEventArgs e
                )
1159        {
1160            move = true;
1161            index_pattern = 7;
1162        }
1163
1164        private void pictureBox8_MouseUp(object sender, MouseEventArgs e
                )
1165        {
1166            move = true;
1167            index_pattern = 8;
1168        }
1169
1170        private void pictureBox9_MouseUp(object sender, MouseEventArgs e
                )
1171        {
1172            move = true;
1173            index_pattern = 9;
1174        }
1175
1176        private void pictureBox10_MouseUp(object sender, MouseEventArgs
                e)
1177        {
1178            move = true;
1179            index_pattern = 10;
1180        }
1181
1182        private void button2_Click_1(object sender, EventArgs e)
1183        {
1184            Color color = getColor();
1185            alive = new SolidBrush(color);
```

```
1186            ALIVE = ColorHandler.fromColorToInt(color);
1187            initMosaics();
1188            PBAutomataSimulator.Invalidate();
1189        }
1190
1191        private void BTNDeadCells_Click(object sender, EventArgs e)
1192        {
1193            dead = new SolidBrush(getColor());
1194            initMosaics();
1195            PBAutomataSimulator.Invalidate();
1196        }
1197
1198        private void BTNGrid_Click(object sender, EventArgs e)
1199        {
1200            grid = new Pen(getColor());
1201            PBAutomataSimulator.Invalidate();
1202        }
1203
1204        private void CBPoints_CheckedChanged(object sender, EventArgs e)
1205        {
1206            if (CBPoints.Checked)
1207                CHHistogram.Series[0].ChartType = System.Windows.Forms.
                        DataVisualization.Charting.SeriesChartType.Point;
1208            else
1209                CHHistogram.Series[0].ChartType = System.Windows.Forms.
                        DataVisualization.Charting.SeriesChartType.Column;
1210        }
1211
1212        private void generatePatternsToolStripMenuItem_Click(object
                sender, EventArgs e)
1213        {
1214            try
1215            {
1216                string value = Microsoft.VisualBasic.Interaction.
                        InputBox("Write the dimension of the matrix", "
                        Generate patterns", "", 0, 0);
1217                //Generating patterns
1218                if (Convert.ToInt16(value) <= 4)
1219                    generatePatterns(Convert.ToInt16(value));
1220                else
1221                    MessageBox.Show("I'm sorry but the maximum size it's
                            4 :c");
1222            }
1223            catch (Exception ex) {
1224                Console.WriteLine("An exception has occured capturing
                        text to generate patterns " + ex);
1225            }
1226        }
1227
1228        private void BTNSavePatterns_Click(object sender, EventArgs e)
1229        {
1230            if (CBPatternRecognition.Checked)
1231            {
```

```
1232                    for (int i = 2; i <= 4; i++)
1233                    {
1234                        storePatterns(i);
1235                    }
1236                }
1237                else
1238                    MessageBox.Show("No se han guardado datos hasta ahora");
1239            }
1240
1241            private void NumericGenMem_ValueChanged(object sender, EventArgs
                    e)
1242            {
1243                TAO = (int)NumericGenMem.Value - 1;
1244            }
1245        }
1246    }
```

1.12 Conclusiones

Se ha hecho un análisis utilizando autómatas celulares con memoria y autómatas celulares los cuales no tienen capacidad de recordar, podemos apreciar que el comportamiento de las siguientes generaciones ahora no solo dependerá de las reglas que son aplicadas con cada generación del autómata celular, es decir a las reglas del juego de la vida de John Conway, sino que ahora cada τ generaciones es necesario hacer una "recapitulación" de lo almacenado durante ese periodo de tiempo que se ha establecido, y determinar mediante a las reglas adicionales implementadas el comportamiento del sistema en la siguiente generación, además es posible notar que aún que utilicemos la misma configuración inicial, el sistema se comporta diferente si el valor de τ es un número par o impar, tanto aplicando regla de mayoría, minoría como paridad se comportan completamente diferente al variar la cantidad de generaciones de las cuales recordaremos algo.

1.13 Referencias

1 "Autómata celular", Es.wikipedia.org, 2018. [Online]. Available: `https://es.wikipedia.org/wiki/Aut%C3%B3mata_celular.[Accessed:28-Nov-2018]`

2 G. Juárez Martínez, A. Adamatzky and R. Alonso-Sanz, "On the Dynamics of Cellular Automata with Memory", ResearchGate, 2015. [Online]. Available: `://www.researchgate.net/publication/279287348_On_the_Dynamics_of_Cellular_Automata_with_Memory.[Accessed:28-Nov-2018]`