Computabilidad y Complejidad:

Automatas celulares

Por: Ramon Ruiz Dolz
Salvador Martí Román

Autómatas Celulares (1)

Para esta primera parte de la segunda práctica, se nos pide el diseño de un módulo Mathematica que recibiendo como parámetros de entrada una configuración inicial, una regla y el número de configuraciones a calcular, simula al autómata celular correspondiente.

Para ello hemos diseñado varios módulos que se encargaran de hacer cada uno una tarea diferente. El primero, ReglaFormato lo que hace es, a partir de un número decimal, crea las tripletas que representan la regla, mediante los números de Von Neumann. Esto se consigue facilmente, ya que podemos convertir el numero a binario con IntegerDigits y después solo queda ir añadiendo cada número al final de los números de Von Neumann.

```
ReglaFormato[regla_] := Module[{rule, listbin, VNnum, i},
  rule = regla;
  listbin = Reverse[IntegerDigits[rule, 2, 8]];
  VNnum = {{0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1}, {1, 0, 0},
{1,
     0, 1}, {1, 1, 0}, {1, 1, 1}};
  For[i = 1, i <= Length[listbin], i++,
   AppendTo[VNnum[[i]], listbin[[i]]];
   ];
  Return[VNnum];
  ]
```

Por otra parte tenemos la transición, es decir el cambio producido sobre la configuración inicial por cada número de configuración a calcular. El mecanismo de la transición es sencillo, este consiste en ir haciendo matching con la regla y sustituyendo el valor por el nuevo. Luego rotamos la configuración, hasta haber realizado esto con todas las celdillas.

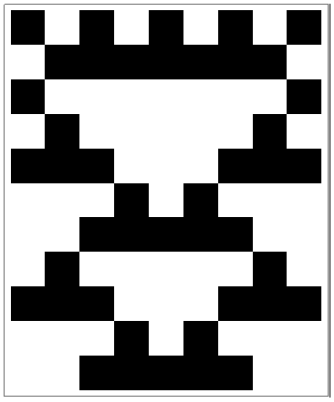
```
Transicion[regla_, estado_] := Module[{ini, fin, i, val, reg,
len},
  ini = estado;
  reg = ReglaFormato[regla];
  fin = {};
  len = Length[ini];
  For[i = 0, i < len, i++,
   val = Cases[reg, {ini[[len]], ini[[1]], ini[[2]], _}];
   val = Flatten[val];
   AppendTo[fin, val[[4]]];
   ini = RotateLeft[ini];
   ];
  Return [fin];
  ]
```

Ramon Ruiz Dolz
Salvador Marti Roman

Finalmente tenemos el módulo que se encargará de realizar el cómputo de las nuevas configuraciones. Este tampoco tiene mucha complicación ya que simplemente itera por tantas configuraciones como haya que calcular, esto es uno de los parámetros que se le pasan, y va llamando al módulo transición que va generando las siguientes configuraciones. Estas se almacenan en una variable y finalmente se realiza un ArrayPlot para observar el resultado.

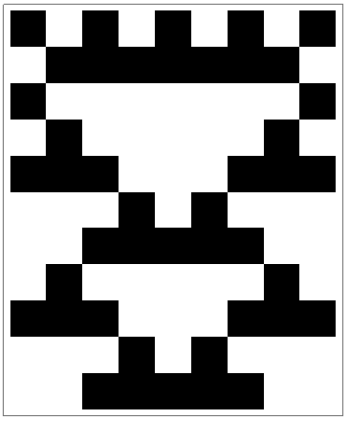
```
ACunidim[estado_, regla_, numtrans_] := Module[{t, final, reg, i,
aux},
  t = numtrans;
  aux = estado;
  final = {estado};
  reg = regla;
  For[i = 0, i < t, i++,
   aux = Transicion[reg, aux];
   AppendTo[final, aux];
   ];
  ArrayPlot[final]
  ]
```

```
ACunidim[{1, 0, 1, 0, 1, 0, 1, 0, 1}, 54, 10]
```



Este es el resultado de aplicar nuestro módulo al estado inicial {1, 0, 1, 0, 1, 0, 1, 0, 1}, la regla 54, 10 veces. Como podemos observar, es idéntico al de abajo, el módulo provisto directamente por mathematica.

```
ArrayPlot[CellularAutomaton[54, {1, 0, 1, 0, 1, 0, 1, 0, 1}, 10]]
```



Este es el resultado de aplicar el módulo CellularAutomaton provisto por Mathematica al estado inicial {1, 0, 1, 0, 1, 0, 1, 0, 1}, la regla 54, 10 veces.

Ramon Ruiz Dolz
Salvador Marti Roman

Autómatas Celulares (2)

1. Implemente un módulo en Mathematica que, tomando como entrada un AFD proporcione como salida un AC unidireccional unidimensional que acepte el mismo lenguaje que el AFD mediante entrada paralela.

Este primer ejercicio pide la conversión de un AFD a un AC unidireccional unidimensional equivalente, para ello se ha programado el siguiente módulo, que a partir de un AFD determinado, generará S, f y S+. Para S tendremos tanto los estados como los símbolos del AFD, para S+ tan solo será coger los estados finales del AFD, y para la regla se realizarán las combinaciones de estados, símbolos y se añadirán las transiciones del AFD, obteniendo así un AC={S,f,S+} totalmente equivalente al AFD original.

```
Act1[automata_] := Module[{i, k, est, sim, tran, regla, fin, AC},
   est = automata[[1]];
   sim = automata[[2]];
   tran = automata[[3]];
   fin = automata[[5]];
   regla = {};
   AC = {};
   For[i = 1, i <= Length[est], i++,
    For[k = 1, k <= Length[est], k++,
      AppendTo[regla, {est[[i]], est[[k]], est[[k]]}];
      ];
    ];
   For[i = 1, i <= Length[sim], i++,
    For[k = 1, k <= Length[sim], k++,
      AppendTo[regla, {sim[[i]], sim[[k]], sim[[k]]}];
      ];
    ];
   For[i = 1, i <= Length[tran], i++,
    AppendTo[regla, tran[[i]]];
    ];
   AppendTo[est, sim];
   AppendTo[AC, Flatten[Union[est]]];
   AppendTo[AC, Union[regla]];
   AppendTo[AC, Union[fin]];
   Return[AC];
   ];
```

```
automata = {{q, p, r}, {a,b}, {{q, a, p}, {q, b, r}, {p, a, r},
{p, b, q}, {r, a, r}, {r, b, r}}, q, {r}}
```

Ramon Ruiz Dolz
Salvador Marti Roman

Aquí podemos observar la salida que obtenemos al aplicar el ejercicio 1 al autómata finito determinista anterior.

```
Act1[automata]

{{p, q, r, a, b}, {{a, a, a}, {a, b, b}, {b, a, a}, {b, b, b}, {p, a, r},
{p, b, q}, {p, p, p}, {p, q, q}, {p, r, r}, {q, a, p}, {q, b, r}, {q, p, p},
{q, q, q}, {q, r, r}, {r, a, r}, {r, b, r}, {r, p, p}, {r, q, q}, {r, r, r}},
{r}}
```

2. Implemente un módulo Mathematica que, tomando como entrada una cadena arbitraria, un AC que reconoce un lenguaje mediante entrada paralela en tiempo real y un estado de frontera q S, proporcione como salida True si el AC acepta la cadena y False en caso contrario.

Este ejercicio nos pide implementar una aceptora de palabras usando un autómata celular, para ello implementamos dos módulos, un primer módulo que va llamando al siguiente módulo para calcular las configuraciones y al final comprueba si se acepta la palabra o no en función de si la última célula tiene un estado final o no.

```
Act2[automata_, palabra_, q_] := Module[{s, f, S, i, cel, len},
   s = automata[[1]];
   f = automata[[2]];
   S = automata[[3]];
   len = Length[palabra];
   cel = Prepend[palabra, q];
   For[i = 1, i <= len, i++,
    cel = Transicion2[f, cel]
    ];
   Return[MemberQ[S, cel[[len + 1]]]];
   ];
```

El segundo módulo se encarga de calcular las nuevas configuraciones a partir de la configuración actual y la regla. Funciona de una forma muy similar al módulo Transición explicado anteriormente, hace matching de las tripletas y coge el 3er valor para añadirlo a la nueva configuración obteniendo así, al finalizar las iteraciones, la configuración resultado al transitar.

```
Transicion2[regla_, estado_] := Module[{ini, fin, i, val, reg,
len},
  ini = estado;
  reg = regla;
  fin = {};
  len = Length[ini];
  For[i = 2, i <=  len, i++,
   val = Cases[reg, {ini[[i - 1]], ini[[i]], _}];
```

```
   val = Flatten[val];
   AppendTo[fin, val[[3]]];
   ];
  fin = Prepend[fin, ini[[1]]];
  Return [fin];
  ]
```

3. Proporcione una construcción formal para obtener a partir de un AFD, un AC equivalente unidimensional bidireccional (vecindario {-1,0,1}) con entrada paralela. Implemente un módulo en Mathematica que lleve a cabo la construcción propuesta.

Para obtener el AC bidireccional equivalente realizaremos algo similar al primer ejercicio, pero esta vez se tendrá en cuenta la celda de la izquierda también, para ello en cada iteración, además de obtener las tripletas anteriores añadiremos un elemento más y obtendremos cuatripletas con las combinaciones de estados y símbolos posibles.
Una vez obtenida la regla, la construcción del AC es idéntica al primer ejercicio, primero se añade S, la unión de estados y símbolos, después la regla y finalmente los estados finales del ADF, S+.

```
Act3[automata_] :=
 Module[{aux, i, k, j, est, sim, tran, regla, fin, AC},
  est = automata[[1]];
  sim = automata[[2]];
  tran = automata[[3]];
  fin = automata[[5]];
  aux = {};
  regla = {};
  AC = {};
  For[i = 1, i <= Length[est], i++,
   For[k = 1, k <= Length[est], k++,
     For[j = 1, j <= Length[est], j++,
       AppendTo[regla, {est[[i]], est[[k]], est[[j]], est[[k]]}];
       ];
     ];
   ];
  For[i = 1, i <= Length[sim], i++,
   For[k = 1, k <= Length[sim], k++,
     For[j = 1, j <= Length[sim], j++,
       AppendTo[regla, {sim[[i]], sim[[k]], sim[[j]], sim[[k]]}];
       ];
     ];
   ];
  For[i = 1, i <= Length[tran], i++,
   For[k = 1, k <= Length[est], k++,
    aux = Insert[tran[[i]], est[[k]], 3];
    AppendTo[regla, aux];
```

```
    ];
   For[k = 1, k <= Length[sim], k++,
    aux = Insert[tran[[i]], sim[[k]], 3];
    AppendTo[regla, aux];
    ];
   ];
  AppendTo[est, sim];
  AppendTo[AC, Flatten[Union[est]]];
  AppendTo[AC, Union[regla]];
  AppendTo[AC, Union[fin]];
  Return[AC];
  ]
```

4. Implemente un módulo Mathematica que muestre la secuencia de computación que se lleva a cabo en el módulo propuesto en la actividad (2) mediante un diagrama espaciotemporal. (Nota: Utilice las funciones ArrayPlot y ListAnimate explicados en la práctica de Fundamentos Básicos)

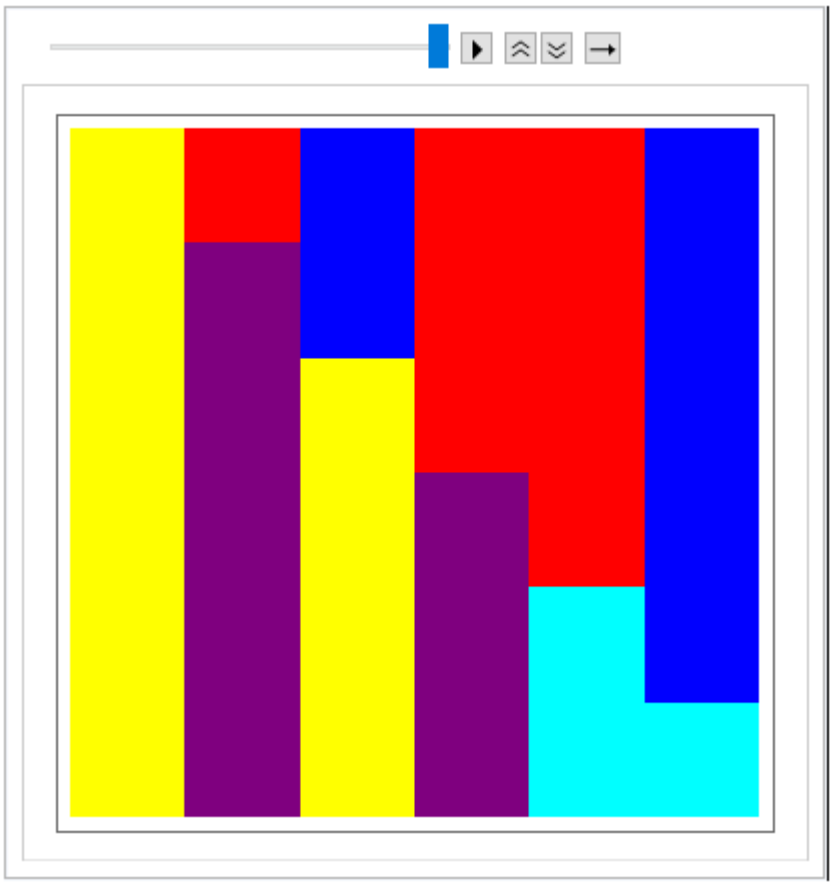
El cuarto y último ejercicio nos pide una representación de las secuencias de computación que realiza el AC, algo similar a lo realizado en la primera parte de esta práctica. Primeramente generamos un frame vacío, a partir del cual comenzará la animación. Tras esto, comenzaremos a iterar y por cada cambio de configuración generamos un frame nuevo, quitando una configuración y añadiendo la nueva. De esta forma, al aplicar ListAnimate obtenemos una animación frame a frame donde podemos observar las secuencias de computación aparecer desde abajo hacia arriba obteniendo al final la imagen inferior.

```
Act4[automata_, palabra_, q_] :=
 Module[{aux, aux2, s, f, S, i, k, cel, len, final, frame},
  s = automata[[1]];
  f = automata[[2]];
  S = automata[[3]];
  len = Length[palabra];
  cel = Prepend[palabra, q];
  aux = {};
  aux2 = {};
  frame = {};
  final = {cel};
  For[k = 0, k <= len, k++,
   AppendTo[aux2, 0];
   ];
  For[i = 0, i <= len, i++,
   AppendTo[aux, aux2];
   ];
  AppendTo[frame,
   ArrayPlot[aux,
    ColorRules -> { a -> Red, b -> Blue, q -> Yellow, p -> Purple,
      r -> Cyan}]];
```

```
  For[i = 0, i <= len, i++,
   aux = Rest[aux];
   AppendTo[aux, cel];
   cel = Transicion2[f, cel];
   AppendTo[final, cel];
   AppendTo[frame,
    ArrayPlot[aux,
     ColorRules -> { a -> Red, b -> Blue, q -> Yellow, p ->
Purple,
       r -> Cyan}]];
   ];
  Return[ListAnimate[frame]]
  ]
```

Aquí podemos observar el último frame de la animación.

```
Act4[autocel, {a, b, a, a, b}, q]
```



Autómatas Celulares (3)

El juego de la Vida

El juego de Vida concebido por J.H. Conway es un ejemplo de cómo dados autómatas celulares con reglas extremadamente simples se pueden crear sistemas verdaderamente complejos. Las reglas son las siguientes:

- Cada celdilla tiene solamente 2 estados:
    - Vivo
    - Muerto
- Se usa la vecindad de Moore. Es decir, se consideran las 8 casillas adyacentes (adyacentes incluidas)
- Las transiciones son las siguientes:
    - Si la celdilla está viva y tiene dos o tres celdillas circundantes vivas, entonces continúa viva en la siguiente etapa, en otro caso muere.
    - Si la celdilla está muerta y tiene exactamente tres celdilla circundantes vivas, entonces vive en la siguiente etapa.

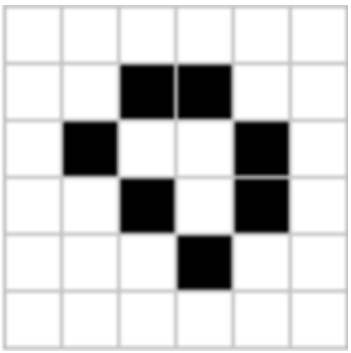
Usando estos estados, reglas y transiciones hay diversas configuraciones tipo que surgen:

- Invariantes: Son subconfiguraciones estables que permanecen inalterables en las siguientes etapas.
- Osciladores: Son configuraciones que exhiben un comportamiento cíclico tal que tras k evoluciones vuelve a producirse la misma subconfiguración. Esto puede producirse en otro lugar ya que el desplazamiento no debe ser necesariamente cíclico.
- Gliders: son osciladores que se desplazan a lo largo de una columna, fila o diagonal. Normalmente se reserva el término glider para aquellos que se desplazan por la diagonal y spaceships al resto.
- Eaters: Son subconfiguraciones sin desplazamiento que eliminan a los gliders que encuentran cuando colisionan de manera apropiada y puede que se recuperen después de cierto número de etapas.

Los guns of gliders: Son osciladores sin desplazamiento que emiten gliders periódicamente.

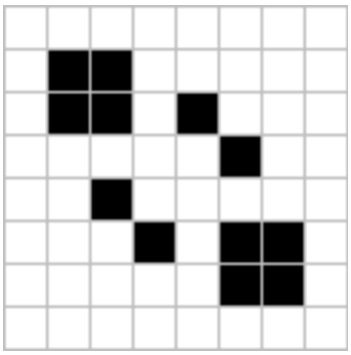
-

1.

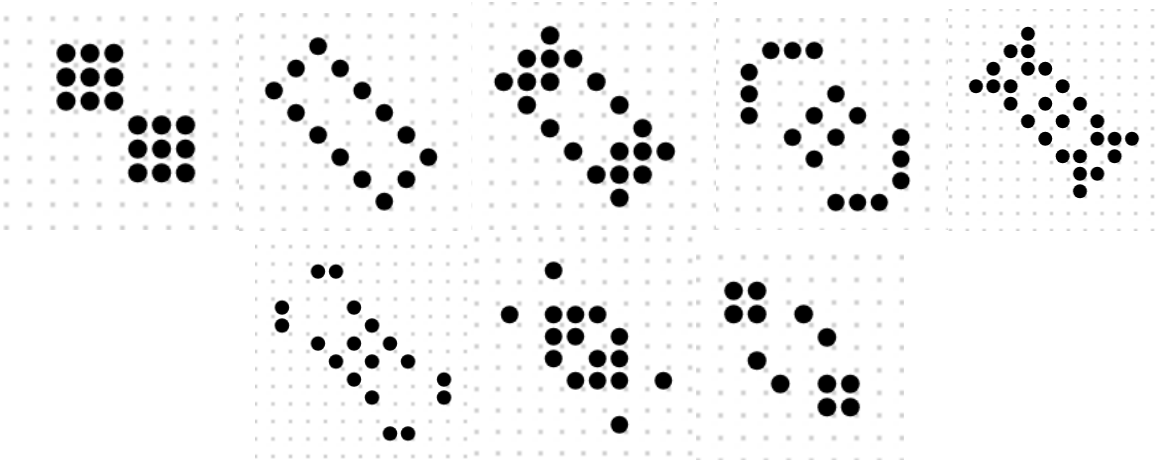


La configuración mostrada en la figura anterior es del tipo invariante esto se debe a que todas las celdillas que empiezan vivas tienen exactamente 2 circundantes vivas manteniendo el estado actual en todas las iteraciones.

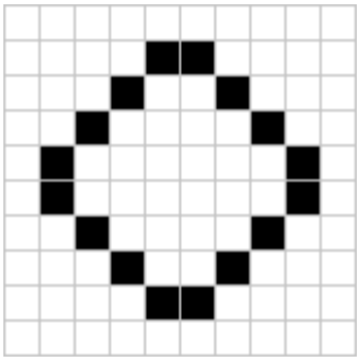
2.



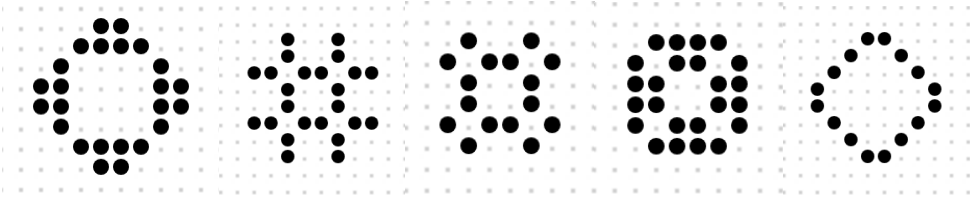
Aunque esta figura aparenta tener 2 secciones del tipo invariable en forma de cuadrados de 2x2 esta configuración se trata de un oscilador de periodo 8 debido a la proximidad entre esos cuadrados y las líneas diagonales. Podemos ver la traza de las configuraciones que toma a continuación:



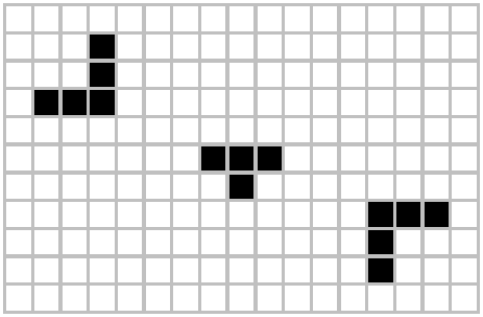
3.



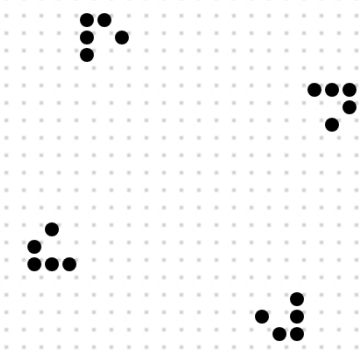
Esta configuración es un oscilador de periodo 5 la traza de las transformaciones se puede ver a continuación:



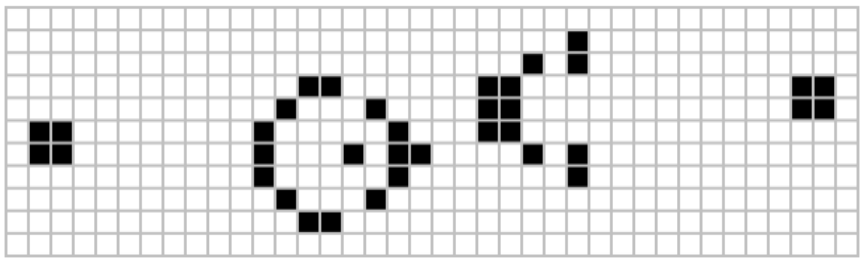
4.



Esta configuración resulta en 4 gliders idénticos pero rotados que se desplazan en direcciones diferentes. Toman la siguiente forma:



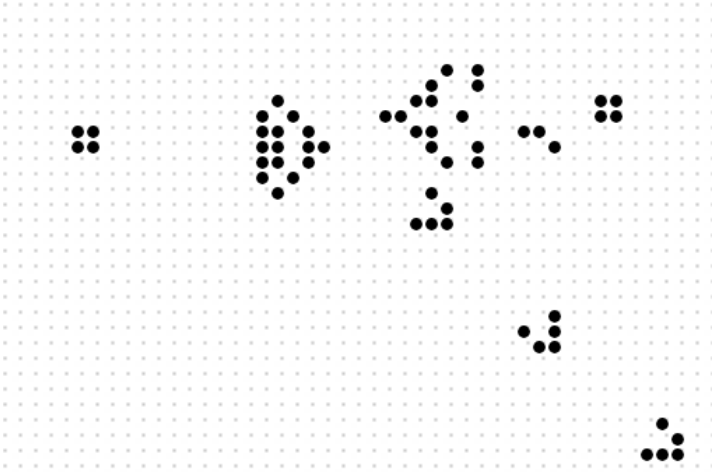
5.



Esta configuración evoluciona a un sistema más complejo que los anteriores. Se pueden identificar varios elementos de diferentes tipos.
Tenemos 2 spaceships osciladores con periodo 30 delimitados por los cuadrados 2x2 que se encuentran a ambos lados. Cuando estos 2 spaceships se acercan a los cuadrados estos cambian de sentido y regresan en la dirección donde vinieron.

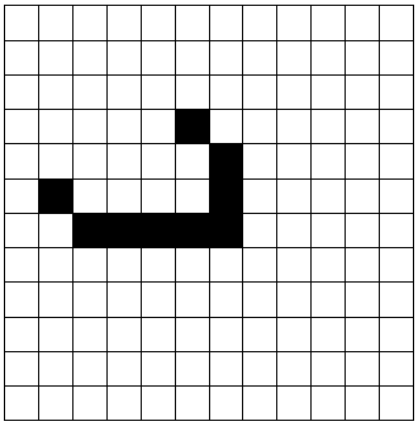
Cuando los eaters chocan estos cambian de sentido una vez más pero emiten un glider que se desplaza por la diagonal inferior derecha. Este bucle nunca acaba.

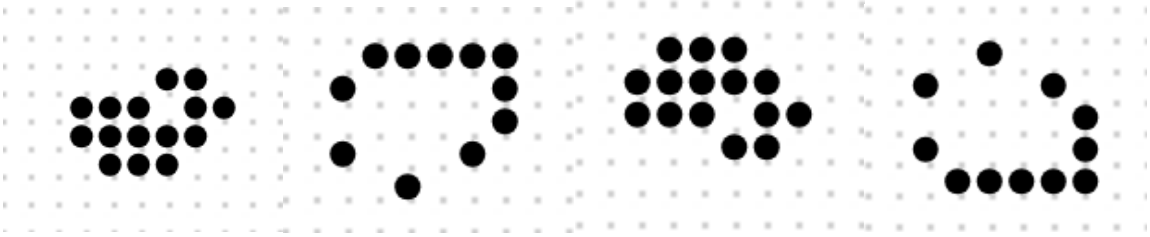
La configuración resultante es un “Guns of gliders”



Ramon Ruiz Dolz
Salvador Marti Roman

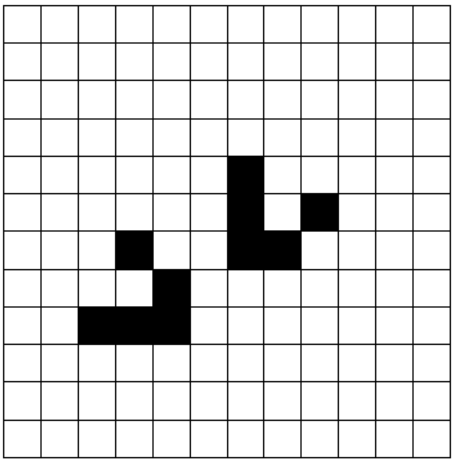
6.



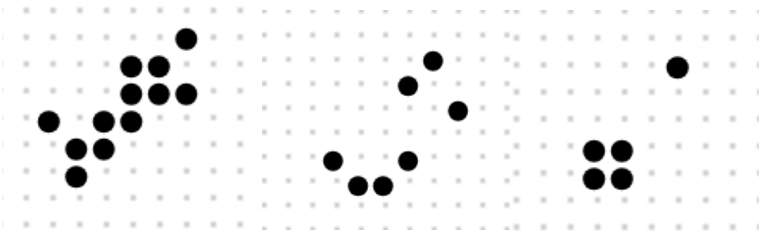


La configuración anterior resulta en un glider que se mueve horizontalmente hacia la derecha alternando su dirección entre las diagonales superior derecha e inferior derecha cancelando así su desplazamiento vertical pero manteniendo el horizontal. Consigue esto oscilando con un periodo 5 entre las configuraciones vistas anteriormente.

7.

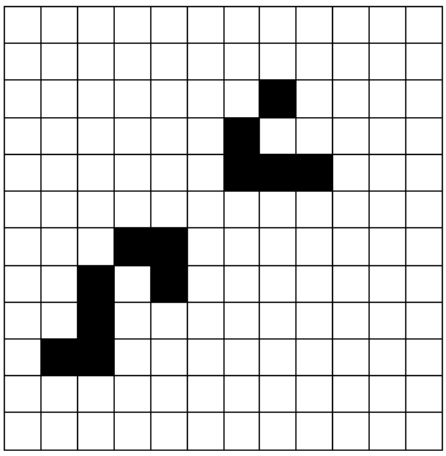


Tras 4 evoluciones la configuración actual encuentra un estado invariante en el cual se mantiene en forma de cuadrado 2x2.



Ramon Ruiz Dolz
Salvador Marti Roman

8.



En esta configuración el glider (la pieza superior derecha) choca con el eater de tal manera que el eater se mantiene y hace desaparecer al glider. Podemos ver la traza a continuación.

Ramon Ruiz Dolz
Salvador Marti Roman