Práctica 2 – Autómatas celulares

Computabilidad y complejidad

Iñaki Diez Lambies y Aitana Menárguez Box

PARTE 1: Fundamentos básicos

Actividad propuesta

A continuación, se muestra el algoritmo implementado en *Mathematica* comentado y explicado paso a paso.

```
In[3]:= AC[des_, regla_, t_] := Module[{lRegla, res, i, aux},
                                  módulo
  lRegla = listaRegla[regla]; (* regla en forma de lista *)
  res = {};
  aux = des; (* aux comienza siendo des *)
  AppendTo[res, aux]; (* se añade la descripción inicial a la solución *)
  añade al final
  (* la primera iteración representar estado inicial *)
  For[i = 1, i ≤ t, i++,
  para cada
   aux = siguienteDes[aux, lRegla]; (* aux es la siguiente descripción *)
   AppendTo[res, aux];
   añade al final
   (* en cada iteracion calcular la descripcion siguiente y representarla*)
  ];
  ArrayPlot[res]
  representación de arreglo
 ]
```

Este código puede dividirse en 3 partes. Primero se calcula la regla en forma de lista, después se calculan las sucesivas descripciones instantáneas del autómata y por último se muestran de forma gráfica.

Por simplicidad del código, se han creado dos submódulos extra que calculan diferentes datos necesarios para la resolución del problema:

- El primero es *listaRegla* que, dada un entero que representa una regla, devuelve la misma regla que se le pasa pero en forma de lista de tal forma que contendrá 8 elementos. Cada uno de ellos será a su vez otra lista de 4 elementos, donde los 3 primeros codificarán las diferentes posibles vecindades de una célula y el último será el estado al cual transicionará la célula que cumpla con dicha vecindad. Para conseguir esto, se cambiará de base decimal a binaria el entero recibido. Cada bit será el siguiente posible estado al que se transicionaría si se cumpliera la vecindad que marca la posición de ese bit.

```
In[2]:= listaRegla[n_] := Module[{b, l, c, d, r, div, i},
                          [módulo
  (* Dado un entero decimal n (número de Wolfram), se devolverá una lista de listas, en la cual cada lista
   es el conjunto de estados de las células de la vecindad y el último elemento el estado que alcanzará la célula en el siguiente instante *)

  b = {}; (* número en binario *)
  l = {{0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1}, {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}}; (* lista con  *)
  c = 0; (* cociente *)
  d = n; (* dividendo *)

  While[ d >= 2 , (* se divide sucesivamente entre 2 para pasar el decimal a binario *)
  [mientras
   div = QuotientRemainder[d, 2]; (* div es la división *)
         [cociente y resto
   r = div[[2]]; (* resto de la división *)
   AppendTo[b, r]; (* en b se van guardando los bits del número binario *)
   [añade al final
   d = div[[1]];
  ];
  AppendTo[b, d];
  [añade al final
  b = PadRight[b, 8]; (* Rellena con 0 hasta tener 8 elementos *)
      [rellena por la derecha
  For[i = 1, i ≤ 8, i++,
  [para cada
   AppendTo[l[[i]], b[[i]]] (* para cada vecindad, se asigna un estado al que transicionar *)
   [añade al final
  ];
  Return [l];
  [retorna
 ]
```

- El segundo se trata de *siguienteDes* que proporciona, partiendo de una descripción instantánea de un autómata celular, la siguiente teniendo en cuenta una regla en específico (calculada anteriormente en forma de lista). Para cada célula y su vecindad, se empareja la célula con el estado al que debe transicionar.

```
siguienteDes[des_, regla_] := Module[{nDes, n, t, q, i},
                              módulo
  n = Length[des]; (* tamaño de la descripción *)
      longitud
  (* hay que tener en cuenta la frontera periódica *)
  (* calcular desde la primera descripción dada *)
  t = Cases[regla, {des[[n]], des[[1]], des[[2]], _}]; (* primera transición *)
      casos
  q = Flatten[t][[4]];
      aplana
  nDes = {};
  AppendTo[nDes, q];
  añade al final
  For[i = 2, i < Length[des], i++, (* se itera hasta calcular n-1 transiciones *)
  para cada          longitud
   t = Cases[regla, {des[[i - 1]], des[[i]], des[[i + 1]], _}]; (* iésima transición *)
       casos
   q = Flatten[t][[4]];
       aplana
   AppendTo[nDes, q];
   añade al final
  ];
  (* calcular la última *)
  t = Cases[regla, {des[[n - 1]], des[[n]], des[[1]], _}]; (* útlima transición *)
      casos
  q = Flatten[t][[4]];
      aplana
  AppendTo[nDes, q];
  añade al final
  Return[nDes];
  retorna
 ]
```

PARTE 2: Reconocedor de lenguajes

Actividad 1

```
Ej1[afd_] := Module[{q, e, d, q0, F, s, f, sp, aux, i },
             módulo
  (* Aislamos los diferentes componentes del AFD *)
  q = afd[[1]]; e = afd[[2]]; d = afd[[3]];
  q0 = afd[[4]]; F = afd[[5]];
  (* Los estados son la unión de los símbolos
   y los estados del AFD al igual que con los
   estados terminales  *)
  s = Union[q, e]; sp = F;
      unión
  (* La función de transición del AFD
   está incluida en la del AC *)
  f = d;
  (* Sacamos el resto de transiciones entre
   estados q *)
  aux = Tuples[{q, q}];
        tuplas
  For[i = 1, i ≤ Length[aux], i++,
  para cada        longitud
   AppendTo[aux[[i]], aux[[i]][[2]]];
   añade al final
   AppendTo[f, aux[[i]]];
   añade al final
  ];
  (* Lo mismo para los estados e *)
  aux = Tuples[{e, e}];
        tuplas
  For[i = 1, i ≤ Length[aux], i++,
  para cada        longitud
   AppendTo[aux[[i]], aux[[i]][[2]]];
   añade al final
   AppendTo[f, aux[[i]]];
   añade al final
  ];
  Return [{s, f, sp}];
  retorna
]
```

En la primera actividad se nos propone implementar un módulo de *Mathematica* que, dado un AFD, proporcione un AC unidireccional unidimensional que acepte lo mismo que aceptaba el AFD original. Para ello hemos desarrollado el siguiente código.

Aprovechando las similitudes entre ambas implementaciones (AFD y AC) y siguiendo las directrices de la práctica, en primer lugar, hemos extraído los diferentes elementos del AFD.

Seguidamente hemos definido los estados *S* como la unión entre los símbolos *E* y los estados *Q* del AFD y los estados finales *F* equivalentes a *S+*.

Por último, hemos creado las diferentes combinaciones de estados para completar así nuestra función de transición entre células vecinas. Todo esto siguiendo la implementación propuesta en la práctica para autómatas celulares unidireccionales.

Actividad 2 y 3

En la actividad 2 se nos pedía la implementación de un algoritmo que, dada una cadena arbitraria, un AC y un estado frontera, se nos devolviera si esta cadena era aceptada o no por el AC dado.

Por otro lado, en el ejercicio 3 se nos pedía mostrar la evolución de estados entre las células implicadas. Por su similitud lo hemos decidido implementar conjuntamente, de forma que comentando una línea puedes decidir cual de los dos resultados quieres esperar que te devuelva.

```
Ej2[x_List, ac_, q_] := Module[{s, f, sp, resu, des, sigDes, aux},
                        módulo
  (* Extraemos sus elementos *)
  s = ac[[1]]; f = ac[[2]]; sp = ac[[3]]; des = x;
  resu = {}; (* Donde se guardan todos las descripciones *)
  AppendTo[resu, des]; (* Añadimos la inicial *)
  añade al final
  sigDes = siguienteDes[Prepend[des, q], f]; (* Siguiente descripcion *)
                        añade al principio
  While[ToString[des] != ToString[ sigDes], (* Mientras no hayan dos iguales *)
  mient··· convierte a cadena de··· convierte a cadena de caracteres

   des = sigDes;
   sigDes = siguienteDes[Prepend[sigDes, q], f];
                         añade al principio
   AppendTo[resu, sigDes];
   añade al final
  ];
  ArrayPlot[resu, ColorRules -> {a → Red, b → Blue, q → Yellow, p → Purple, r → Green}] (* Se muestra la evolución *)
  representación de ar··· reglas de color  rojo  azul  amarillo  púrpura  verde
  (*
  Return[MemberQ[sp, des[[-1]]] ](* Si todos los estados de la descripción son estados finales, devolver True *)
  *)

 ]
```

En este código se realiza la implementación donde, como de forma habitual, en primer lugar, extraemos sus características. Seguidamente añadimos su primer estado a una lista auxiliar que nos servirá para poder observar la evolución si fuera deseado. Calculamos seguidamente la siguiente descripción instantánea (teniendo en cuenta su estado frontera) y realizamos un bucle donde vamos calculando todas las posibles hasta que lleguemos a un estado equivalente al anterior, lo que significa que ha terminado la computación.

```
siguienteDes[des_, f_] := Module[{nDes, i, t},
                          módulo
  nDes = {};
  For[i = 2, i ≤ Length[des], i++,
  para cada        longitud
   t = Flatten[Cases[f, {des[[i - 1]], des[[i]], _}]];
       aplana      casos
   AppendTo[nDes, t[[3]]]
   añade al final
  ];
  Return[nDes]
  retorna
 ]
```

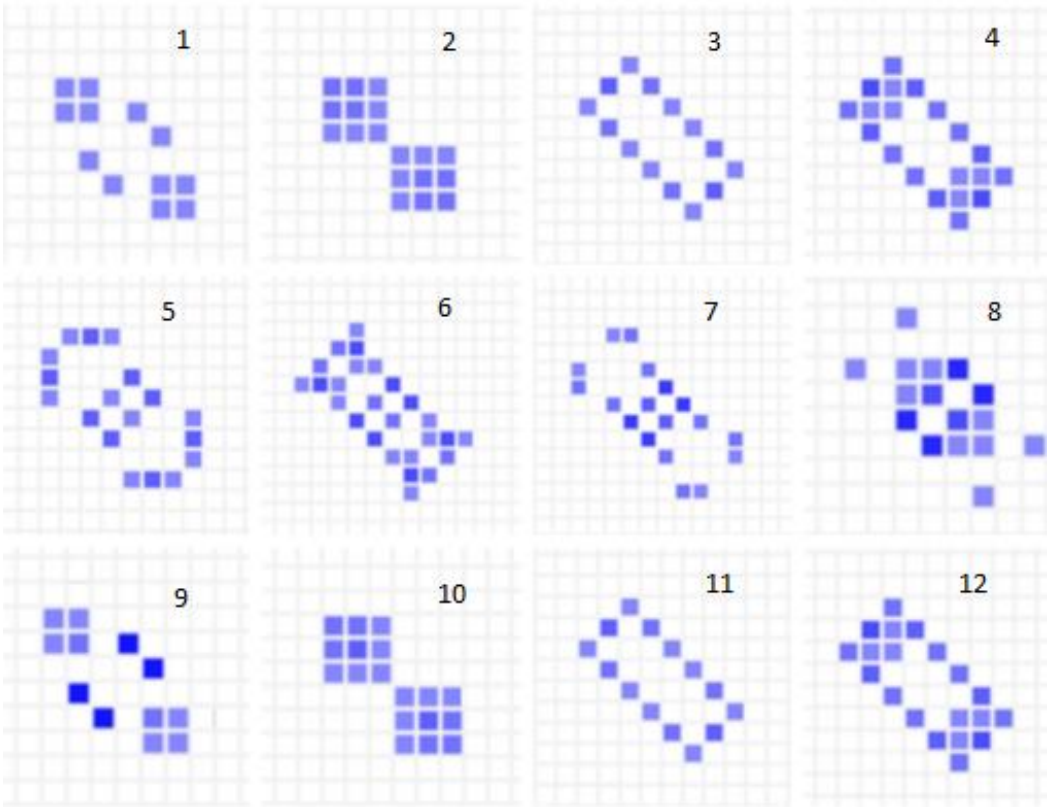
Si queremos seleccionar un ejercicio u otro deberemos de intercambiar el comentario entre las dos últimas líneas de código.

Para realizar el cálculo de la siguiente transición se ha diseñado un módulo adicional que tiene como objetivo devolver la descripción consecuente a una, dada esta y su función de transición. Como podemos observar simplemente se realiza un bucle por todas las ocurrencias de la cadena encontrando el nuevo estado respecto al de la célula y su vecino a la izquierda.

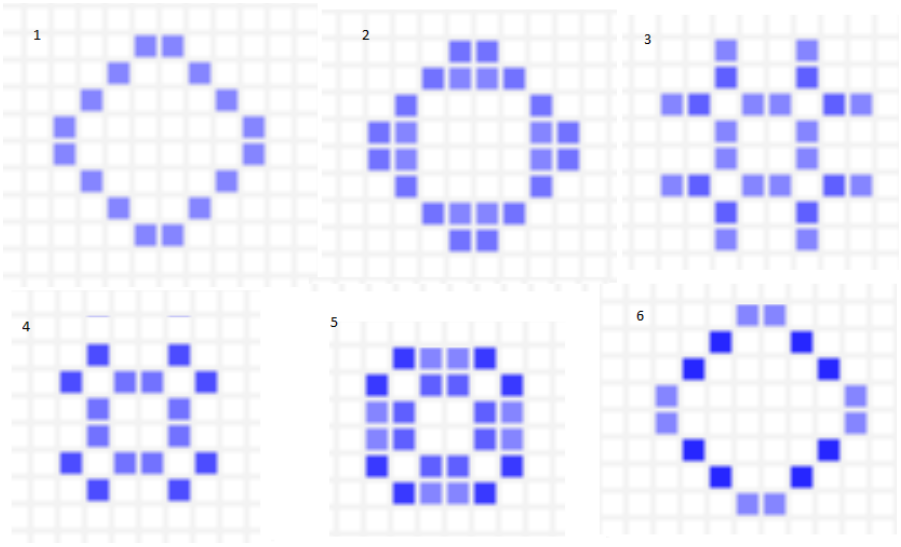
PARTE 3: El juego de la vida

1. Se trata de una configuración invariante. Permanece inalterable en todas sus etapas siguientes. Esto ocurre porque todas las celdillas vivas tienen dos o tres celdillas circuncidantes vivas.

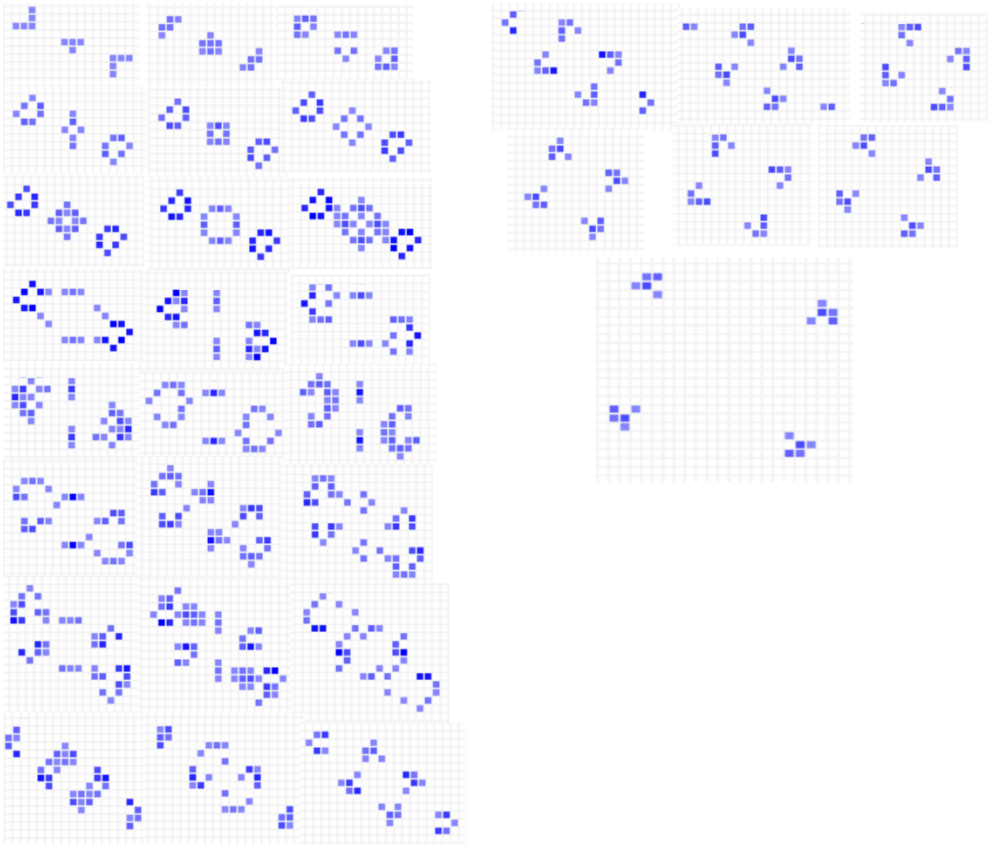
2. Se trata de un oscilador. Presenta un comportamiento cíclico permanente con un período igual a 8, ya que cada 8 etapas, se vuelve a reproducir la misma subconfiguración. Esta configuración no se desplaza.



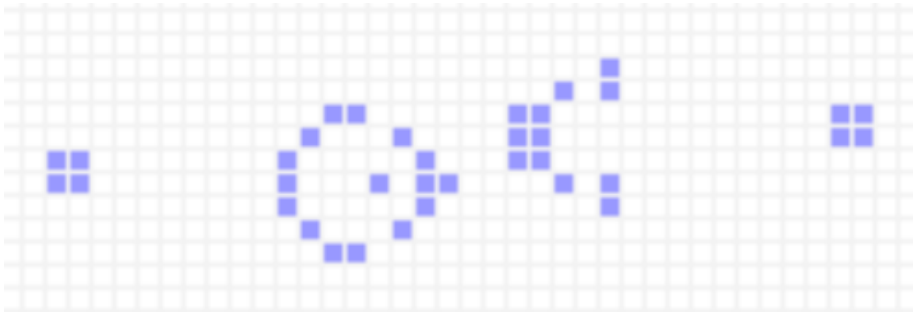
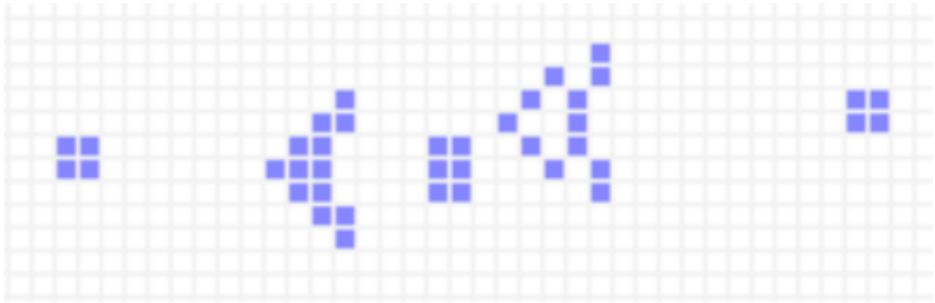
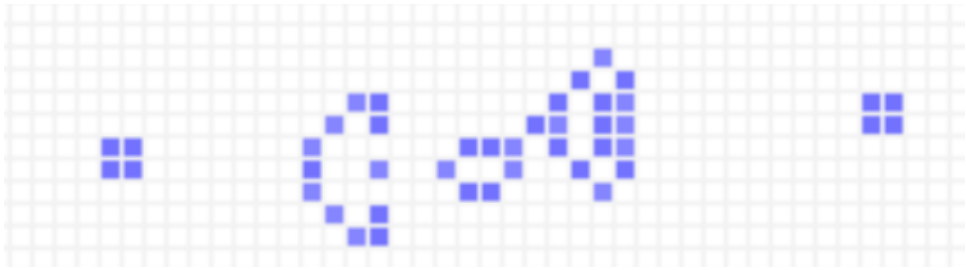
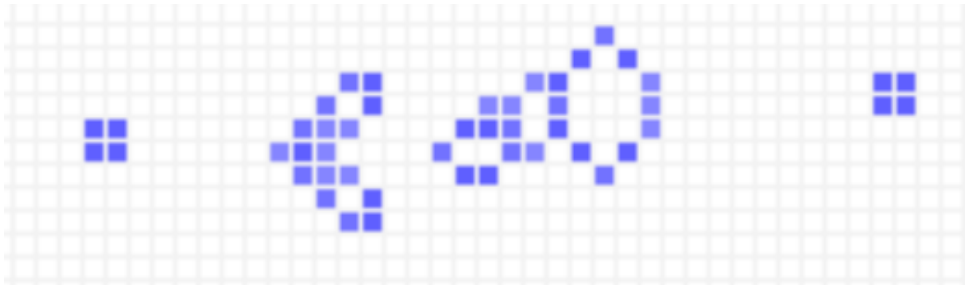
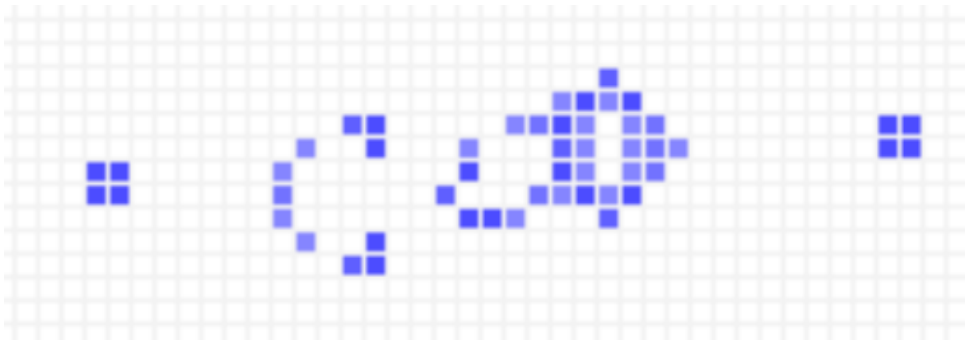
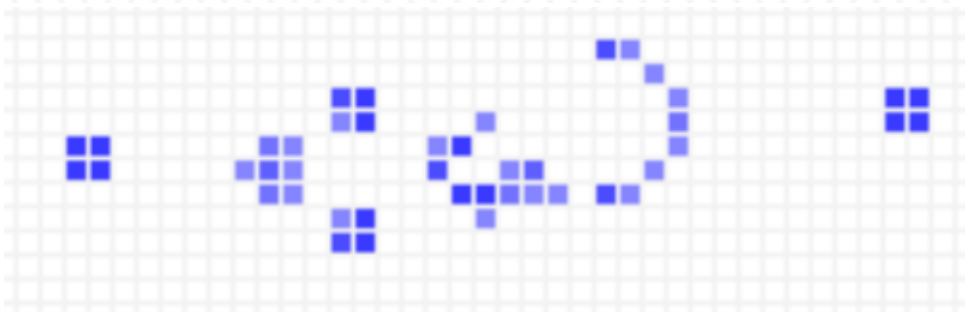
3. Se trata de un oscilador de nuevo. En esta caso tiene un periodo de 6 ya que, tras 6 pasos, vuelve al la misma subconfiguración.

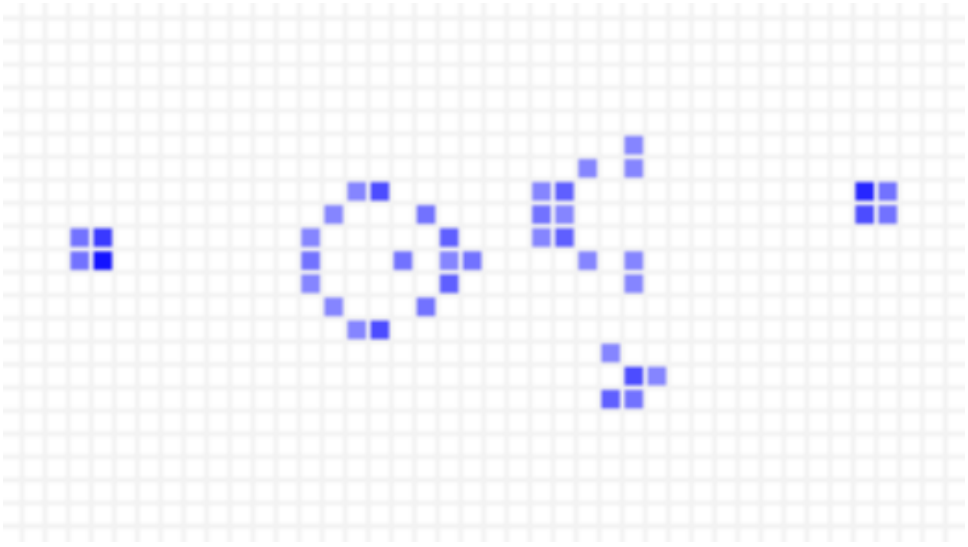


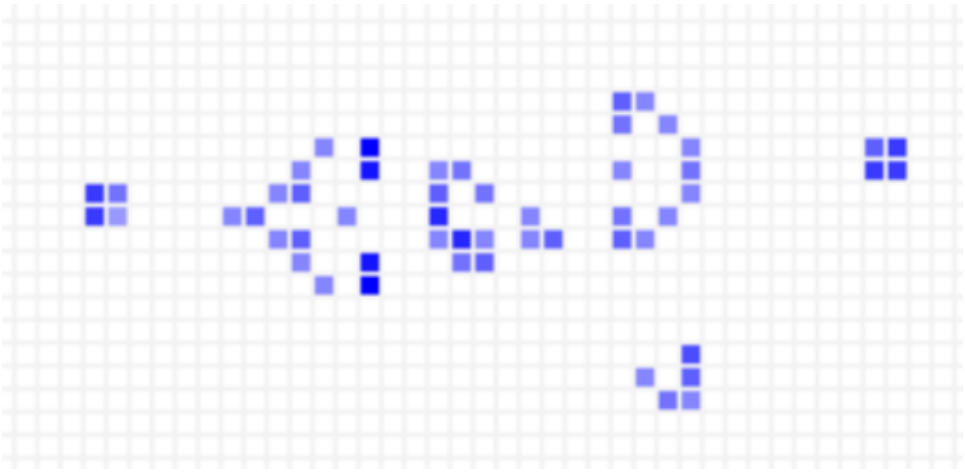
4. En este caso nos encontramos con un generador de 4 gliders. No genera de forma periódica diversos sino que, después de diversas transiciones, acaba por converger en 4 gliders que se disparan en direcciones opuestas.

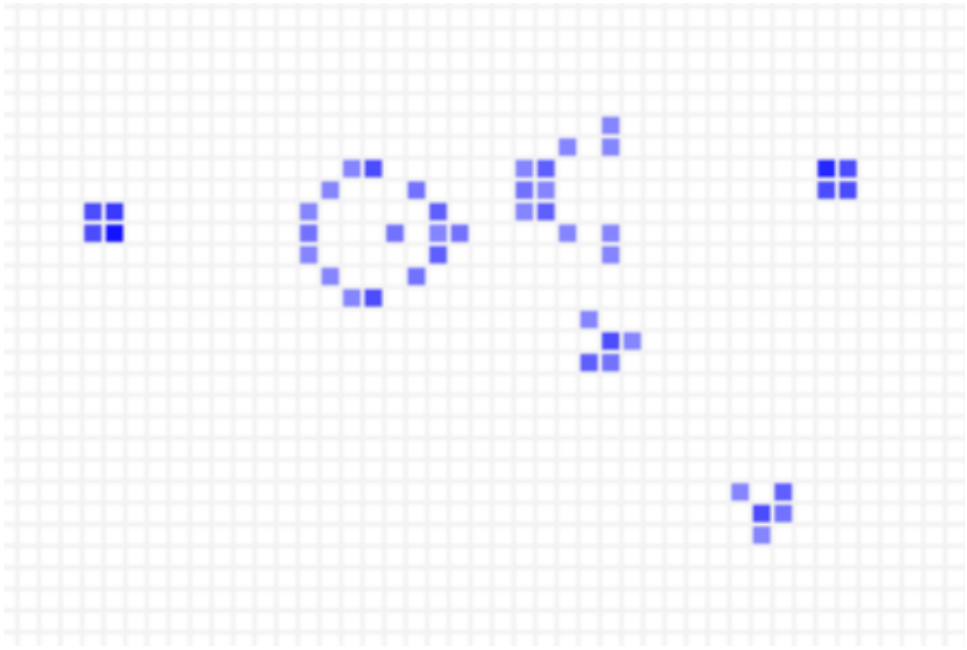


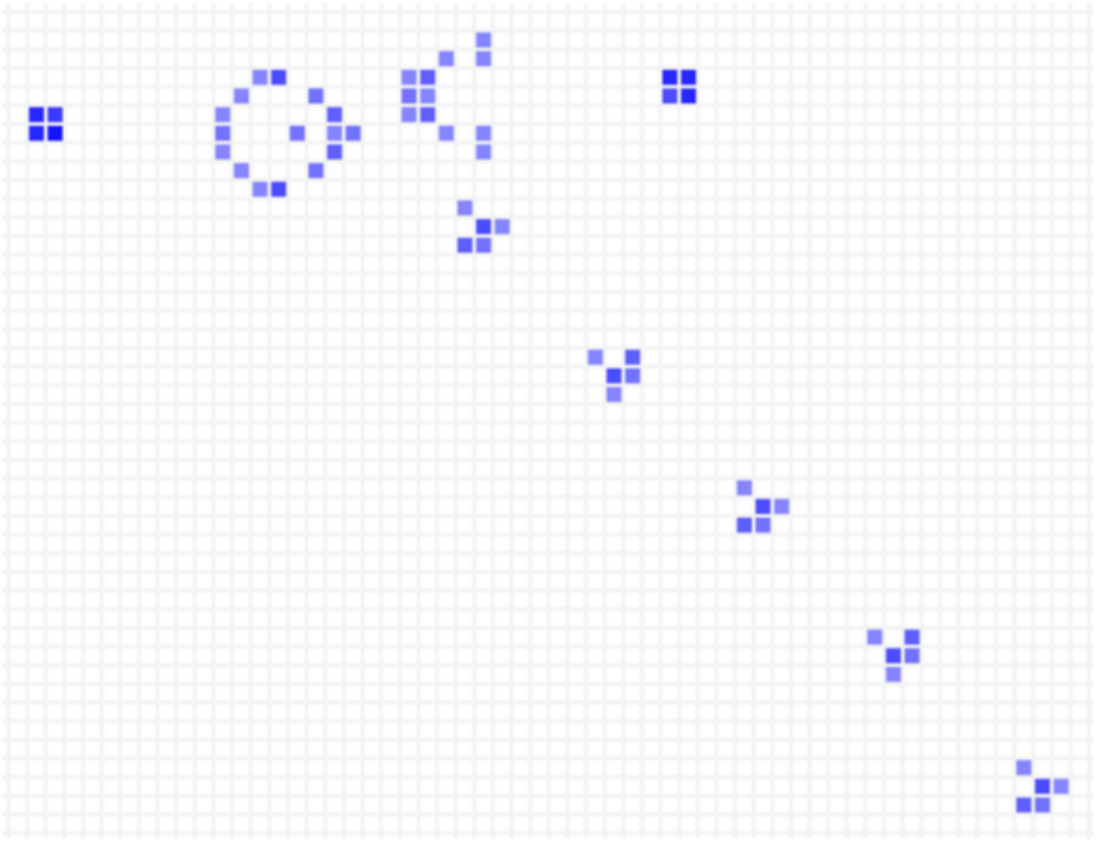
5. Se trata de un gun of gliders. Es un emisor de gliders que emite periódicamente (en este caso, cada 30 configuraciones) un glider. Así, se ve que en la generación 30, se vuelve a repetir la configuración inicial pero con un glider que está siendo emitido. Al cabo de muchas transiciones, en la generación 150, se han emitido 5 gliders. Esta emisión es ilimitada.

1. 
2. 
3. 
4. 
5. 
6. 

7. 

8. 

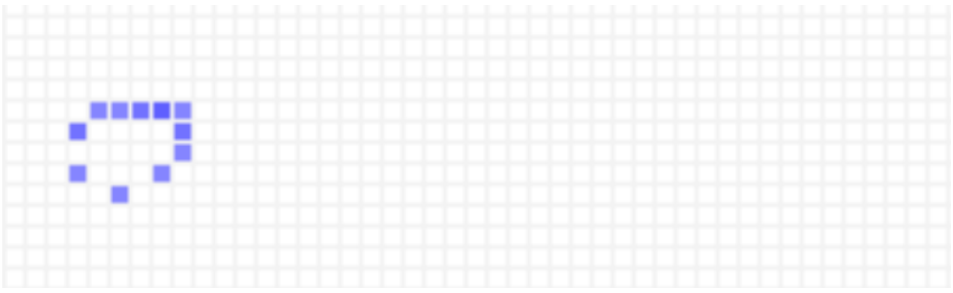
9. 

10. 

6. Se trata de un glider horizontal, un oscilador con desplazamiento hacia la derecha (también llamado spaceship). Si periodo es de 4; cada 4 transiciones se repite la configuración, pero desplazada a la derecha una celda.

0. 

1. 

2. 

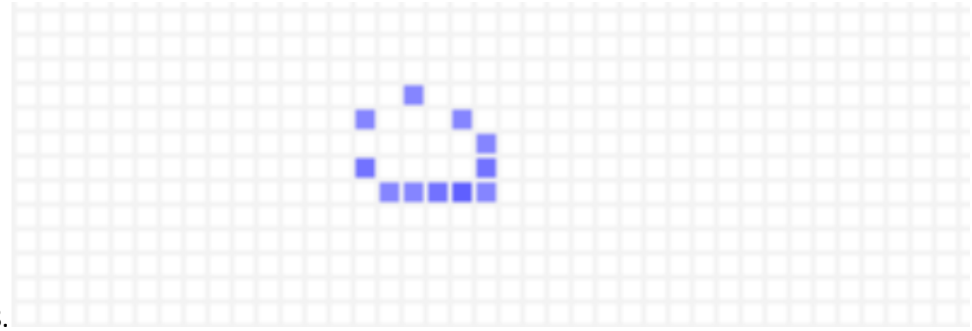
3. 

4. 

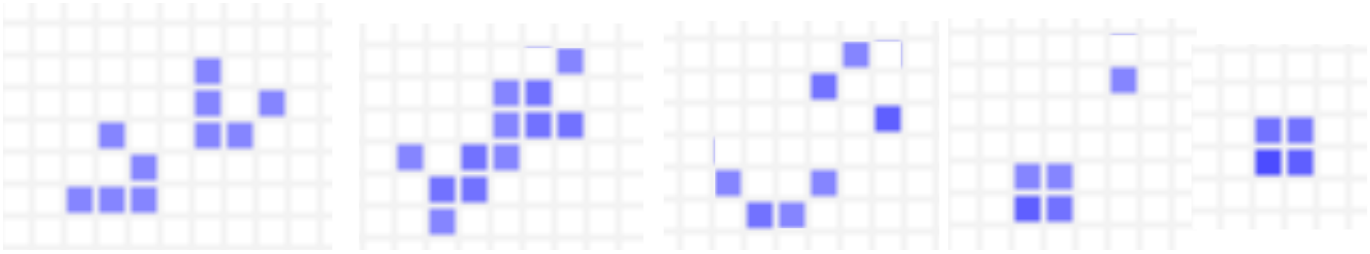
5. 

6. 

7. 

8. 

7. En este caso nos encontramos con una forma que después de 4 transiciones acaba por convertirse en un invariante.



8. En este caso nos encontramos un aniquilador. Es decir, una configuración sin desplazamiento que es capaz de devorar los gliders con los que se encuentran. Podemos observar en las transiciones este proceso.