Notas de la clase de 18 mayo, 2016

- **El juego LIFE.**

  Las reglas del juego se resumen por el código S23/B3: una celda ocupada ("viva") sobrevive a la siguiente generación si tiene 2 o 3 vecinos, y una celda vacia se llena en la siguiente generación (un oranismo "nace", o *born*), si tiene 3 vecinos (exactamente).

  Aquí está el código que usamos en la clase (organizado un poco):

```
clc; close all;
bs=50;              % board size
B=randi(2,bs)-1;    % board
A=zeros(bs+2);      % board surrounded by zeros
r_off= [-1, -1, 0, 1, 1,  1,  0, -1];   % row offset
c_off=[ 0,  1, 1, 1, 0, -1, -1, -1];    % column offset
gen=1;               % generation counter
while true           % end with control-C
    A(2:end-1,2:end-1)=B;
    imagesc((1:bs)+0.5,(1:bs)+0.5,~B);
    colormap(gray);
    axis equal
    title(['generation=' num2str(gen)])
    C= zeros(bs);   % counts the number of neighbors of each cell in B
    for j=1:8
      C= C+A(r_off(j)+(2:end-1),c_off(j)+(2:end-1));
    end
    B=(B & C==2)| C==3;  % the S23B3 rule
    pause(.1)
    gen=gen+1;
end
```

- Aqui está otra opción, muy elegante...

```
clc;clear all;
B = randi(2, 50) - 1; % board
while true      %use control C to stop
   spy(B,30);
   C = conv2(B, [1 1 1; 1 0 1; 1 1 1], 'same'); % count neighbors
   B = double((B & C == 2) | C == 3); % apply S23B3 rule
   pause(.1)
end
```

  El comando `conv2` (convolución) se refiere a una operación matemática muy interesante y útil, pero no vamos a entrar al tema en este curso.

- Aqui están 2 patrones iniciales interesantes (ambos crecen indefinitivamente):

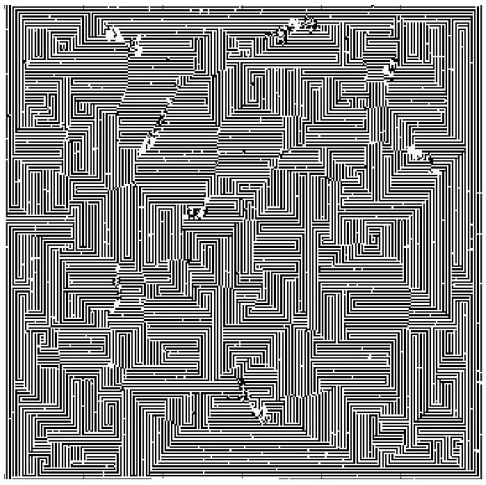
```
1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1

0 0 0 0 0 0 1 0
0 0 0 0 1 0 1 1
0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0
```

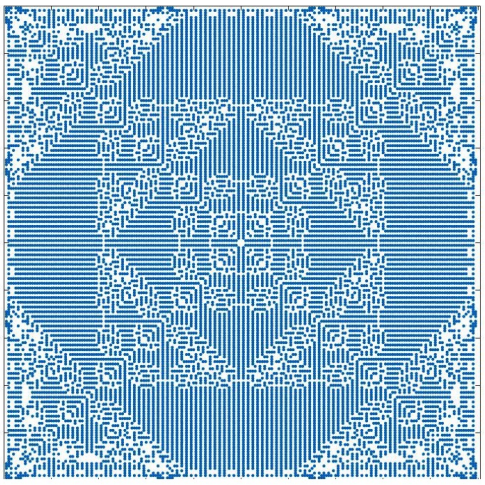
- Hay mucho material en internet sobre el juego y sus variaciones. El artículo de Wikipedia es bueno y tiene ligas. En Matlab existe el command `life` que da una simulacion. En el sitio `www.bitstorm.org/gameoflife/standalone/` se puede bajar un programa de Java, cuya ventaja es que se puede cargarle muy facilmente configuraciones iniciales desde la pagina `www.bitstorm.org/gameoflife/lexicon/`, simplemente arrastrando las configuraciones al tablero del programa.

Tarea num. 14

1. Modificar el programa de la clase para explorar otras reglas de LIFE, diferentes al S23B3. Por ejemplo, S123B13, o S12B1 (empezando con una sola celda en el centro).



S123B13



S12B1

2. Escribir una funcion `insert_figure(filename,bs,i,j)` capaz de leer un patron inicial de LIFE de un archivo de texto, e insertarlo en una matriz de zeros en el lugar (i, j). Para leer el archivo la opción más simple me parece es el comando `load` (ver la documentacion). El comando `load` pone el contenido de archivo leido en una matriz (ver ejemplo abajo).

```
function B=insert_figure(filename,bs,i,j)
% Produces a square matrix B of 0's, reads a pattern of 0's and 1's (an array)
% from a file named 'filename' and inserts it into B so that the upper left corner
% of the pattern goes to B(a,b).
% Important: The text in filename should be a table of 0's and 1's separated by spaces.

fig=load(filename);
. . .
etc
```

   Luego usas esta función dentro de un script de LIFE. Por ejemplo, digamos que queremos correr un "glider" en nuestro programa de LIFE. Ponemos en el archivo `glider` (archivo de texto, lo editas con editor de texto, como bloc de notas en Windows) el patron siguiente

```
0 1 0
1 0 0
1 1 1
```

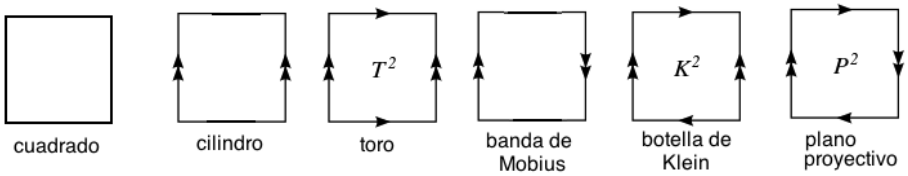
   (los 0's y 1's separados por espacios!) y luego lo podemos leer e insertar en una matriz grande de ceros en un script, digamos así

```
B=insert_figure('glider',50,5,45);
```

3. Modificar el script de LIFE para simular el juego sobre una dona (un "toro"). Esto es, imaginamos que las orillas verticales y horizontales están pegadas entre si. De este modo, *toda* celda del tablero, incluso las de las orillas, tiene 8 celdas vecinas. Por ejemplo, si el tablero es una matriz B de 10×10, entonces los vecinos nuevos de $B(1, 5)$ son $B(10, 4), B(10, 5)$ y $B(10, 6)$; los vecinas nuevos de una esquina, digamos $B(1, 1)$, son $B(1, 10), B(2, 10), B(10, 10), B(10, 1)$ y $B(10, 2)$. Prueba tu programa con varias configuraciones iniciales, como el "glider"., asegurando que funciona correctamente (sobre todo en las esquinas).

   Sugerencia: una manera sencilla de hacerlo es usar la matriz A de nuestro código de la clase, agregando en sus orillas los elementos correctos. Por ejemplo, si ponemos en la 1era columna de A los elements de la última columna de B, con el comando `A(2:end-1,1)=B(:,end)`, entonces efectivamente los elementos de la última columna de B se vuelven vecinos de los elementos de la 1era columna. Así hay que seguir llenando todas las orillas de la A. No se te olvide las esquinas! Por ejemplo, `A(1,1)=B(end,end)`.

4. (Opcional) Repetir el inciso anterior para las superficies que se obtiene pegando las orillas de un cuadrado:



5. (Opcional) Simular el juego de LIFE sobre una tablero con celdas triangulates o hexagonales. Por ejemplo, la regla B2S34 para hexagonal LIFE parece interesante; ver
   `http://davidsiaw.github.io/blog/2014/11/21/hexlife/`