El código Processing® de abajo (https://processing.org) -del cual no me siento particularmente orgulloso- implementa el juego de la vida de Conway (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life), un juego uni-personal que es un referente en teoría de la computación (por ejemplo, en https://www.ics.uci.edu/~welling/teaching/271fall09/Turing-Machine-Life.pdf aparece una implementación de una máquina de Turing en el juego de la vida).

1 Lea el primer capítulo de "The Lifebox, the Seashell, and the Soul" titulado "computation everywhere" y modifique el código para reflejar la "Vichniac Vote Rule". Corra su código con diferentes imágenes iniciales y capture la imagen final después de algunos cientos de iteraciones. ¿aparece algún patrón? Documente sus hallazgos y repórtelos en la primera parte de un documento pdf.

2 Lea los artículos "Wicked Problems - Jay Rosen – In the US, rising health care costs are classic case of a wicked problem. No "right" way to view it" https://edge.org/response-detail/11091) y "The World is Unpredictable - Rudy Rucker – Even if the world is as deterministic as a computer program, you still can't predict what you're going to do" (https://edge.org/response-detail/10171). Como practicante del área de seguridad de la información, escriba un ensayo de una página con sus propias conclusiones de los artículos y de su código.

El código… Ustedes sabrán excusarme.

```
/*
Conway game of life...

             NW   N   NE
             W         E
             SW   S   SE

Rules:
1. Death from isolation: Each live cell with less than two live neighbors dies
in the next generation
2. Death from overpopulation: Each cell with four or more live neighbors dies
in the next generation
3. Birth: Each dead cell with exactly three live neighbors comes to life in the
next generation
4. Survival: Each live cell with two live neighbors survives in the next
generation

*/

int[][] grid, futureGrid;  // current state and next generation
int iterations = 0;

void setup() {
  size(800, 500);
  frameRate(8);
  grid = new int[width][height];
  futureGrid = new int[width][height];

  /* para llenado automático, lo que va por dentro
  float density = 0.3 * width * height;  // densidad inicial... que tan denso
es al principio
  for (int i = 0; i < density; i=i+1) {
    grid[int(random(width))][int(random(height))] = 0;  // en 0 para que no
ponga nada, en 1 para que llene aleatoriamente algunos
```

```
  } // end for() del llenado inicial de la vaina

  */

  /* Voy a hacer un glider
  grid[1][1] = 0;
  grid[2][1] = 1;
  grid[3][1] = 0;
  grid[1][2] = 0;
  grid[2][2] = 0;
  grid[3][2] = 1;
  grid[1][3] = 1;
  grid[2][3] = 1;
  grid[3][3] = 1;
  */

  // este código imprime un patrón de 28 "live" pixels in a straight line that
achieves infinite growth
  grid[401][250] = 1;
  grid[402][250] = 1;
  grid[403][250] = 1;
  grid[404][250] = 1;
  grid[405][250] = 1;
  grid[406][250] = 1;
  grid[407][250] = 1;
  grid[408][250] = 1;

  grid[409][250] = 0;

  grid[410][250] = 1;
  grid[411][250] = 1;
  grid[412][250] = 1;
  grid[413][250] = 1;
  grid[414][250] = 1;

  grid[415][250] = 0;
  grid[416][250] = 0;
  grid[417][250] = 0;

  grid[418][250] = 1;
  grid[419][250] = 1;
  grid[420][250] = 1;

  grid[421][250] = 0;
  grid[422][250] = 0;
  grid[423][250] = 0;
  grid[424][250] = 0;
  grid[425][250] = 0;
  grid[426][250] = 0;

  grid[427][250] = 1;
  grid[428][250] = 1;
  grid[429][250] = 1;
  grid[430][250] = 1;
  grid[431][250] = 1;
  grid[432][250] = 1;
  grid[433][250] = 1;

  grid[434][250] = 0;

  grid[435][250] = 1;
  grid[436][250] = 1;
  grid[437][250] = 1;
  grid[438][250] = 1;
  grid[439][250] = 1;

  background(255,255,255);

  for (int x = 1; x < width-1; x=x+1) {
    for (int y = 1; y < height-1; y=y+1) {
       if(grid[x][y] == 1) {
         set(x, y, color(255));
       } else {
         set(x, y, color(0));
       } // end if then else
    } //end for()
  } // end for()

  saveFrame("frames/first.png");  // el primero de todos...

}  // end setup...
```

```
void draw() {
  iterations = iterations + 1;

  for (int x = 1; x < width-1; x=x+1) {
      for (int y = 1; y < height-1; y=y+1) {

  /*   Para "que pase el chorizo"
  for (int x = 0; x < width; x=x+1) {
      for (int y = 0; y < height; y=y+1) {*/

        // Check the number of neighbors (adjacent cells)
        int nb = neighbors(x, y);

        if ((grid[x][y] == 1) && (nb <  2)) {
          futureGrid[x][y] = 0; // Isolation death
          set(x, y, color(0));
        } else if ((grid[x][y] == 1) && (nb >  3)) {
          futureGrid[x][y] = 0; // Overpopulation death
          set(x, y, color(0));
        } else if ((grid[x][y] == 0) && (nb == 3)) {
          futureGrid[x][y] = 1; // Birth
          set(x, y, color(255));
        } else {
          futureGrid[x][y] = grid[x][y]; // Survive
        } // end if-then-else
    } // end for()
  } // end for()

  // Swap current and future grids
  int[][] temp = grid;
  grid = futureGrid;
  futureGrid = temp;

  if(iterations % 1000000 == 0) {
    saveFrame("frames/glider-######.png");
  } // end if

}  // end draw()


// Count the number of adjacent cells 'on'. Esta función "no pasa el chorizo"...
el objeto llega al límite y muere...
int neighbors(int x, int y) {
  return grid[x][y-1]       // North
         + grid[x+1][y-1]   // Northeast
         + grid[x+1][y]     // East
         + grid[x+1][y+1]   // Southeast
         + grid[x][y+1]     // South
         + grid[x-1][y+1]   // Southwest
         + grid[x-1][y]     // West
         + grid[x-1][y-1]   // Northwest
         ;
} // end neighbors()

/*

Para que la figura pase el chorizo utilice mejor la función de abajo...
int neighbors(int x, int y) {
  int north = (y + height-1) % height;
  int south = (y + 1) % height;
  int east = (x + 1) % width;
  int west = (x + width-1) % width;
    return grid[x][north]       // North
           + grid[east][north]  // Northeast
           + grid[east][y]      // East
           + grid[east][south]  // Southeast
           + grid[x][south]     // South
           + grid[west][south]  // Southwest
           + grid[west][y]      // West
           + grid[west][north]  // Northwest
           ;
} // end neighbors()

*/
```