ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/282770018

Diseño de Algoritmos en Matlab para la Solución de un Sudoku

Article *in* IET Computers & Digital Techniques · June 2013

CITATIONS
0

READS
894

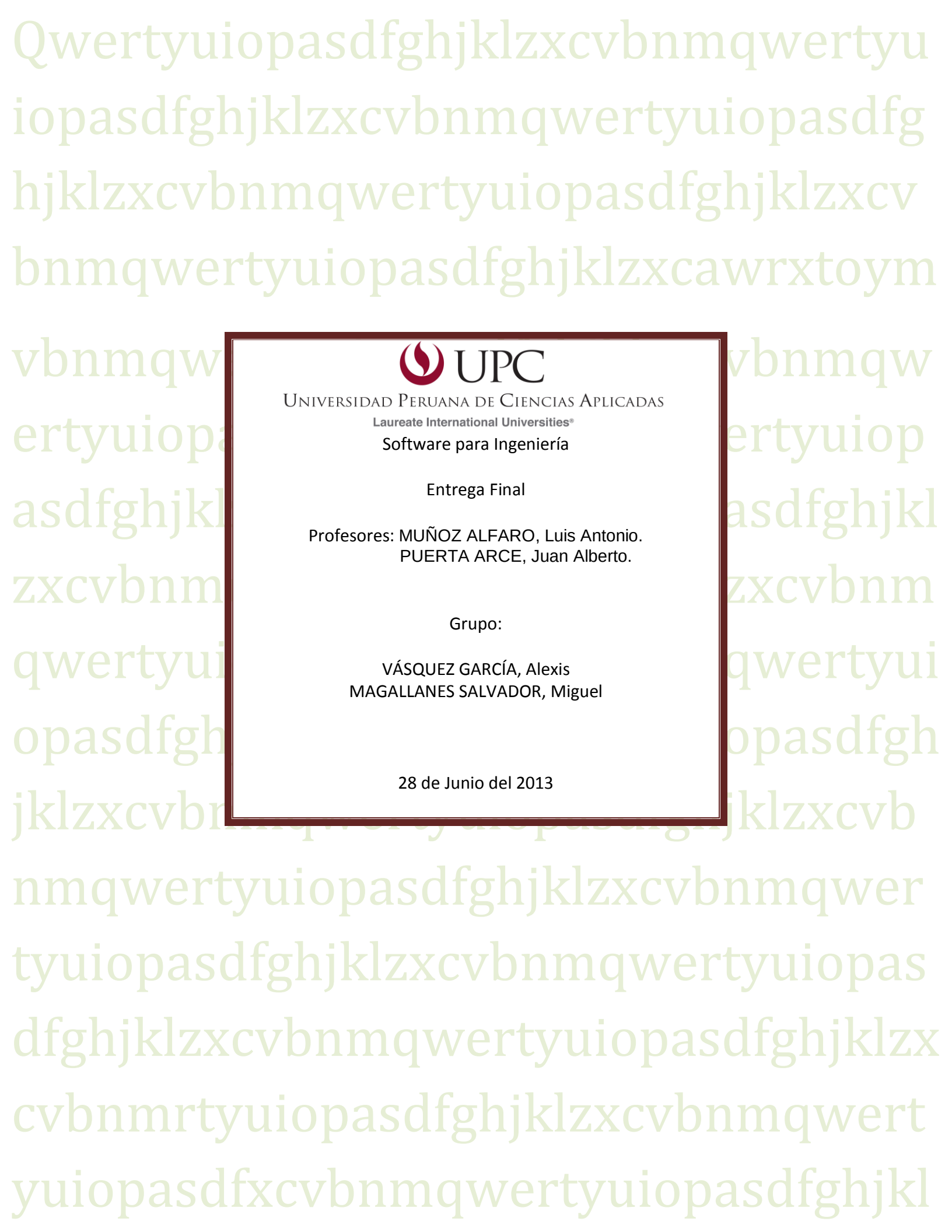
1 author:

Alexis Vásquez García
Universidad Peruana de Ciencias Aplicadas (UPC)
5 PUBLICATIONS **0** CITATIONS

SEE PROFILE

All content following this page was uploaded by Alexis Vásquez García on 12 October 2015.

The user has requested enhancement of the downloaded file.



UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

Laureate International Universities®

Software para Ingeniería

Entrega Final

Profesores: MUÑOZ ALFARO, Luis Antonio.
PUERTA ARCE, Juan Alberto.

Grupo:

VÁSQUEZ GARCÍA, Alexis
MAGALLANES SALVADOR, Miguel

28 de Junio del 2013

SOFTWARE PARA INGENIERÍA

SUDOKU

ÍNDICE

1. **Información sobre el sudoku ……4**
   a) **¿QUÉ ES? ……4**
   b) **¿CÓMO SE JUEGA? ……4**
2. **Proceso de resolución ……5**
   a) **Pseudocódigos ……7**
   b) **Algoritmos ……..8**
   c) **Funciones ……..15**
      - **Primarias ……15**
        - ✓ **ElementoPosicion ……15**
        - ✓ **Howmany ……15**
      - **Para el sudoku ……16**
        - ✓ **BuscaFalta ……16**
        - ✓ **BuscaComun3 ……18**
        - ✓ **BarridoCuad ……19**
        - ✓ **BarridoSud ……..20**
        - ✓ **VerSudoku ……..21**
   d) **Programa: sudoku ……..22**
3. **Pruebas y Resultados ……23**
4. **Bibliografía ……..24**

SUDOKU

¿QUÉ ES?

El sudoku es un pasatiempo que se publicó por primera vez a finales de la década de 1970 y se popularizó en Japón en 1986, dándose a conocer en el ámbito internacional en 2005 cuando numerosos periódicos empezaron a publicarlo en su sección de pasatiempos.

¿CÓMO SE JUEGA?

Este juego se compone de una cuadrícula de 9x9 casillas, que está divida en cuadrículas dentro de la más grande que son de 3x3 casillas. Partiendo de algunos números dispuestos en algunas de las casillas, hay que completar las que estén vacías con dígitos del 1 al 9. No se debe repetir ningún digito en una misma fila, columna o cuadro de 3x3. Un sudoku está bien planteado si la solución es única.

8	3	5	4	1	6	9	2	7
2	9	6	8	5	7	4	3	1
4	1	7	2	9	3	6	5	8
5	6	9	1	3	4	7	8	2
1	2	3	6	7	8	5	4	9
7	4	8	5	2	9	1	6	3
6	5	2	7	8	1	3	9	4
9	8	1	3	4	5	2	7	6
3	7	4	9	6	2	8	1	5

PROCESO DE RESOLUCIÓN

Ahora que entendemos cómo se juega el sudoku, tenemos que buscar un método de resolución para que podamos escribirlo en el programa Matlab y este lo interprete de forma que pueda hacer sudokus fáciles y normales. En este caso, no evaluaremos un sudoku para que resuelva en modo difícil, esto lo explicaremos más adelante.

Como sabemos, necesitamos hacer un script que siga las reglas básicas del juego del sudoku para que este se resuelva. Por lo tanto, lo que debemos tener en cuenta, principalmente, es que al introducir un número dentro del tablero del sudoku, este no se puede repetir en la misma columna, fila o cuadrícula de 3x3. A partir de estas simples reglas tenemos que hacer una serie de pasos para que Matlab pueda interpretar los códigos que escribiremos y pueda resolver el sudoku.

Consideremos las reglas básicas y obtendremos un proceso para la resolución:

1. Dentro del tablero, habrán números del 1 al 9 y ceros.
2. Los únicos que cambiarán de valor son los ceros.
3. El número cambiado en un cero no deberá repetirse en la misma fila, columna o cuadrícula de 3x3.

Luego, seguiremos una serie de pasos recomendadas para la resolución, es una de las más sencillas para poder terminar un sudoku:

1. Evaluamos la casilla donde esté un cero.
2. En esa casilla, evaluamos su fila, columna y cuadrícula de 3x3.
3. Verificamos en cada una los números que faltan.
4. El número faltante que se repita en la fila, columna y cuadrícula de 3x3 será el número que irá en vez del cero.
5. Seguimos este proceso hasta que no quede ningún cero dentro del sudoku.

A continuación, evaluaremos un sudoku fácil para poder hacer nuestros scripts de desarrollo y consideraremos los pasos escritos líneas arriba.

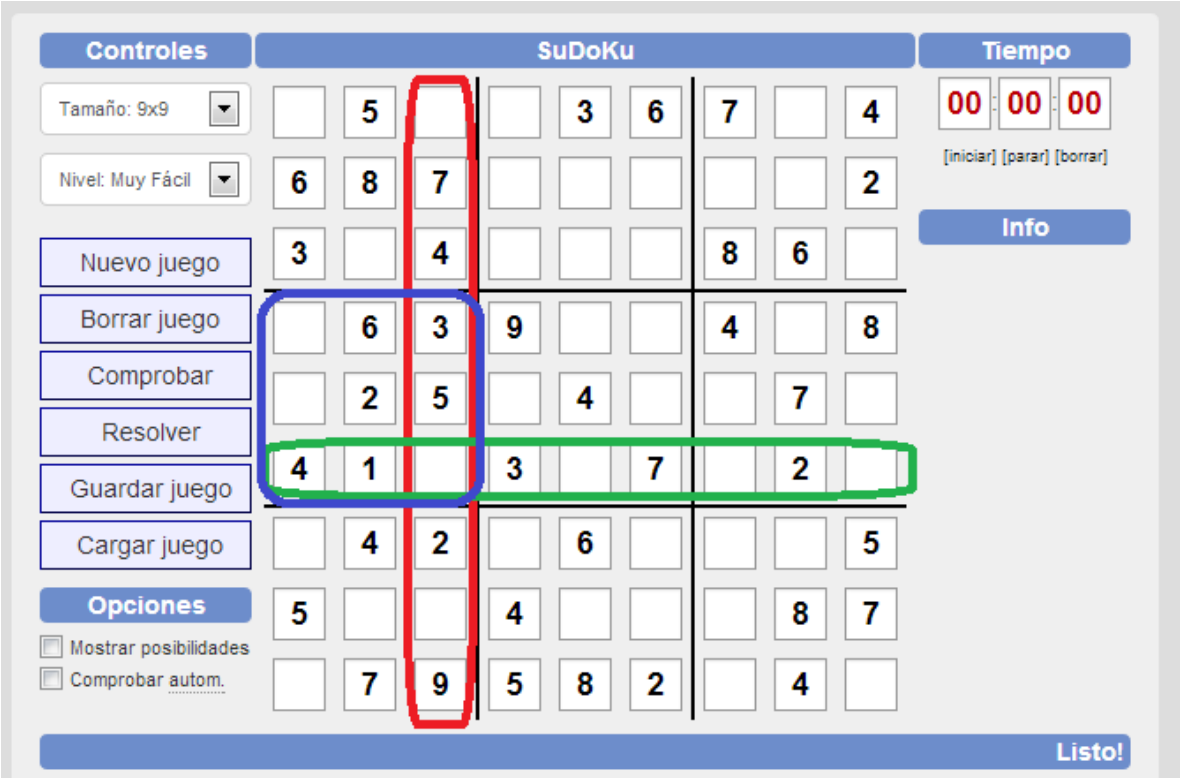


FIGURA 1 Fuente: SUDOKU-ONLINE

En la casilla que se encuentra en la sexta fila y tercera columna, tenemos:

En la fila, escribimos los números faltantes: 5, 6, 8 y 9.
En la columna, los números que faltan son: 1, 6 y 8.
En la cuadrícula de 3x3, faltan los números: 7, 8 y 9.

Luego, el número que se repite en las 3 anteriores es el número "8"; por consiguiente, ese número es el que va en esa casilla.

Observemos que, en este caso, solo hay un número que se repite, en todo caso, si es que hubiese 2 o más, no se podría resolver esta casilla por el método que estamos siguiendo; por consiguiente, se debe analizar la siguiente casilla. Así sucesivamente hasta encontrar una casilla donde solo haya UN NUMERO FALTANTE que se repita.

Entonces, como hemos visto, debemos hacer este proceso para cada casilla en la que haya un cero. Nuestro programa debe evaluar ese caso y detenerse cuando todas las casillas estén resueltas con un número diferente de cero entre el 1 y el 9.

PSEUDOCÓDIGOS

Ahora que ya tenemos las ideas de qué proceso seguir para hacer nuestros scripts de desarrollo para la resolución de un sudoku, escribiremos pseudocódigos para estructurar todo nuestro desarrollo y evaluación del programa.

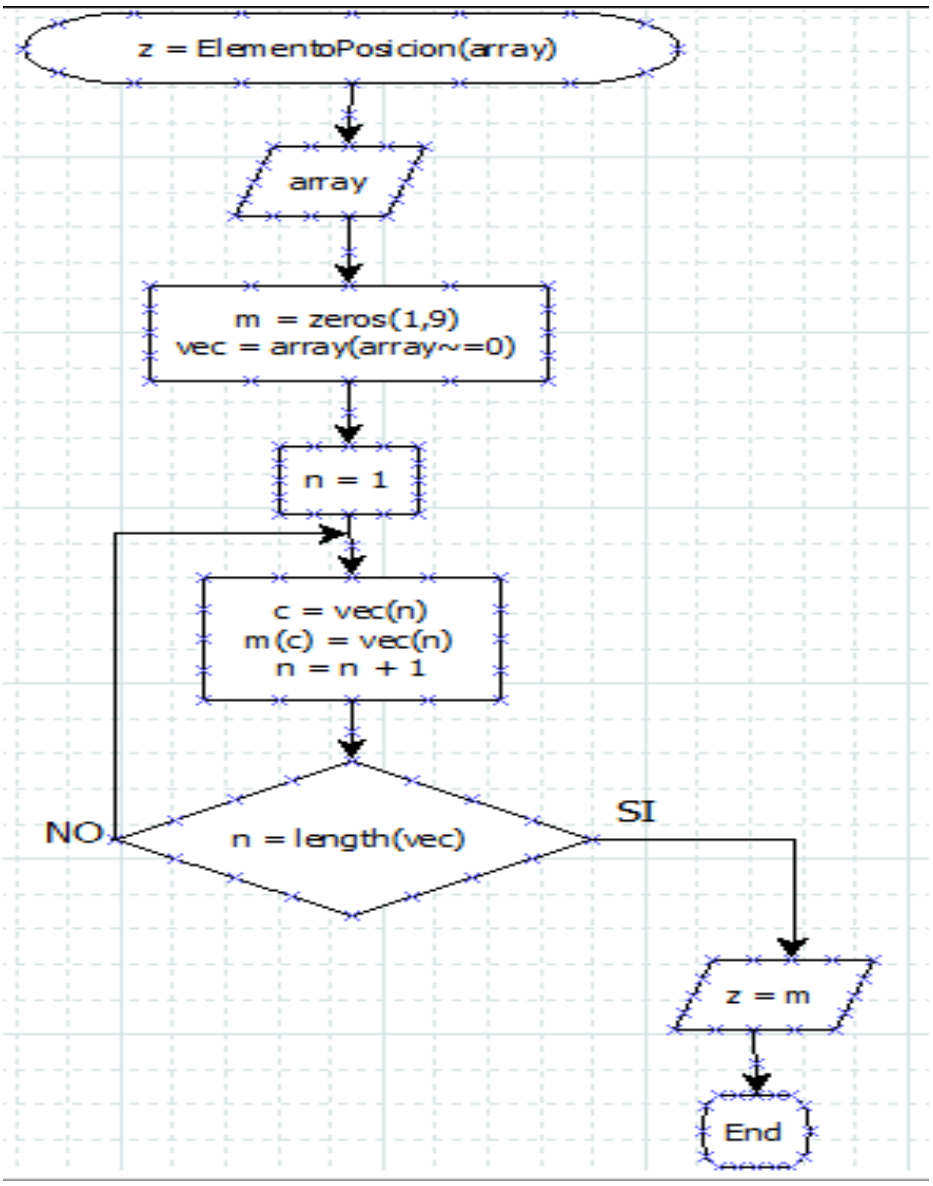
Recopilamos la información necesaria y puntualizamos lo más importante:

1. Generamos un tablero con valores fijos y ceros.
2. Para todas las casillas donde haya un cero se analizará los pasos que siguen a continuación.
   a) De la fila, obtener los números que faltan entre el 1 y 9.
   b) De la columna, obtener los números que faltan entre el 1 y 9.
   c) De la cuadricula de 3x3, obtener los números que faltan entre el 1 y 9.

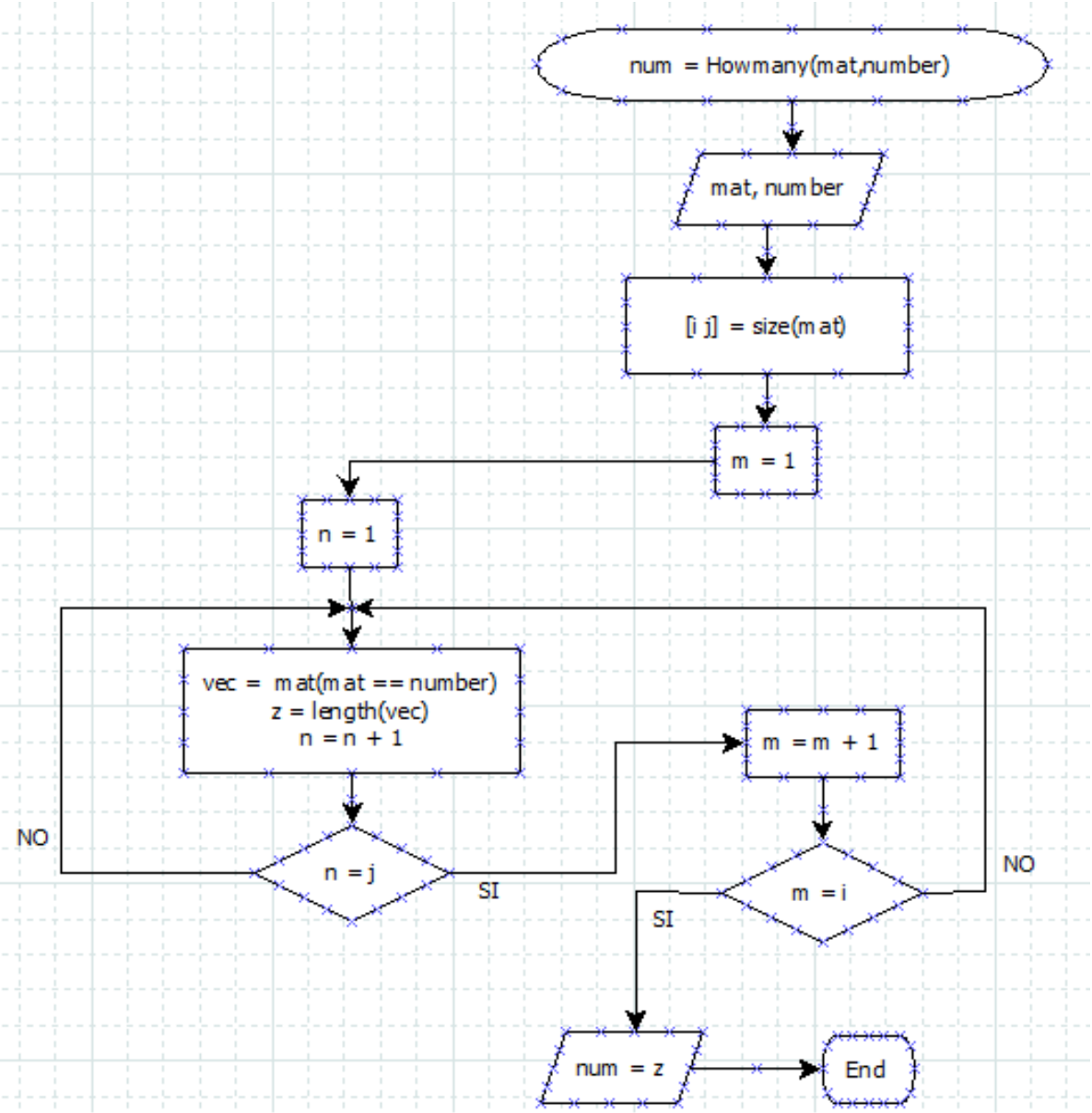
3. El número faltante que se repita en los 3 casos mencionados anteriormente es el que va en la casilla donde se ha analizado.
4. Si existe más de un número faltante que se repita, no se podrá resolver y se procederá a analizar la siguiente casilla donde haya un cero.
5. En caso de que en uno de los barridos que se haga del sudoku, no haya una casilla con un solo número faltante que se repita, entonces el programa deberá mostrar que el sudoku no se puede resolver (al menos no por el método que estamos siguiendo).
6. Mientras aún existan ceros dentro de nuestro tablero, nuestro programa debe seguir evaluando esas casillas.

ALGORITMOS

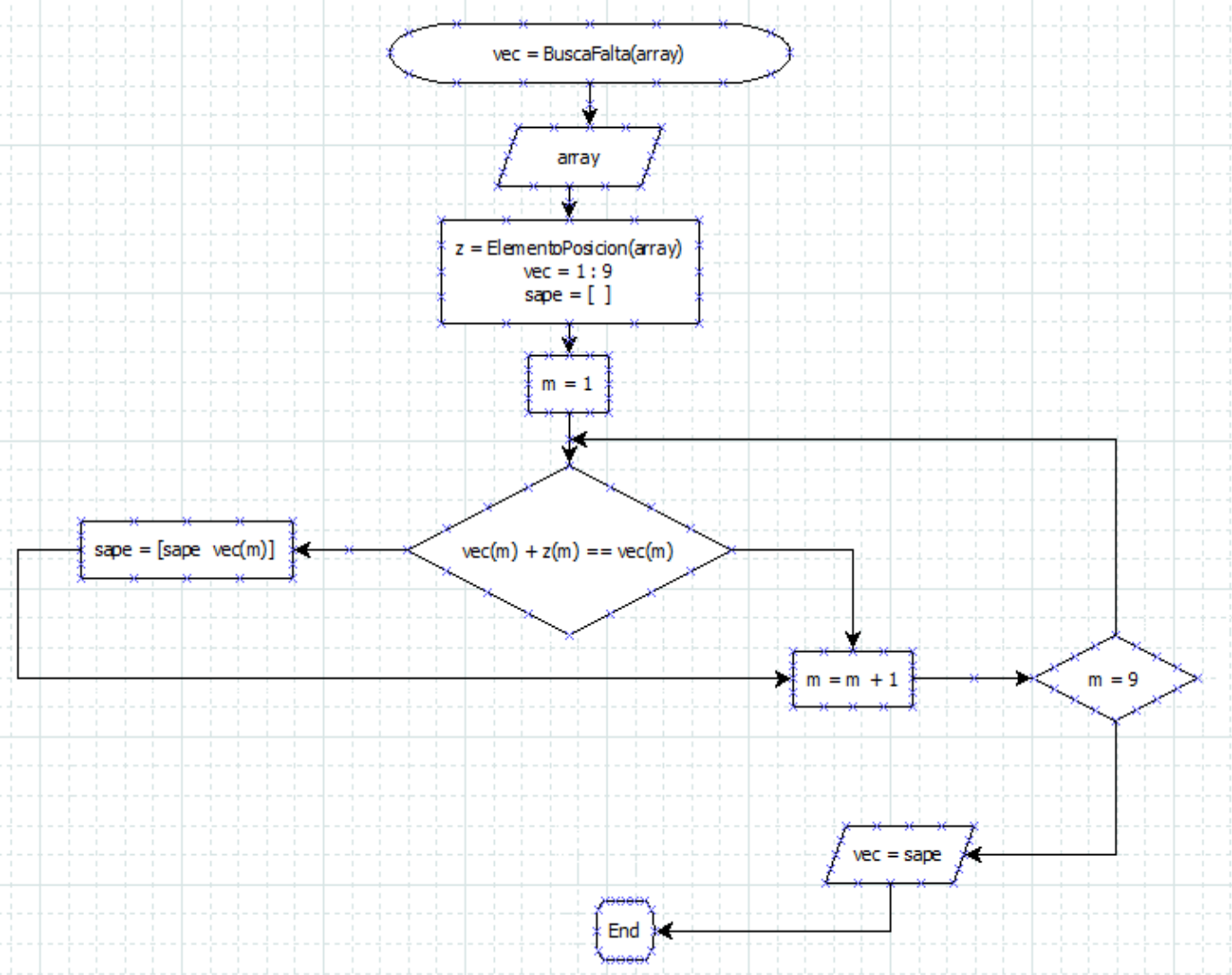
ElementoPosicion



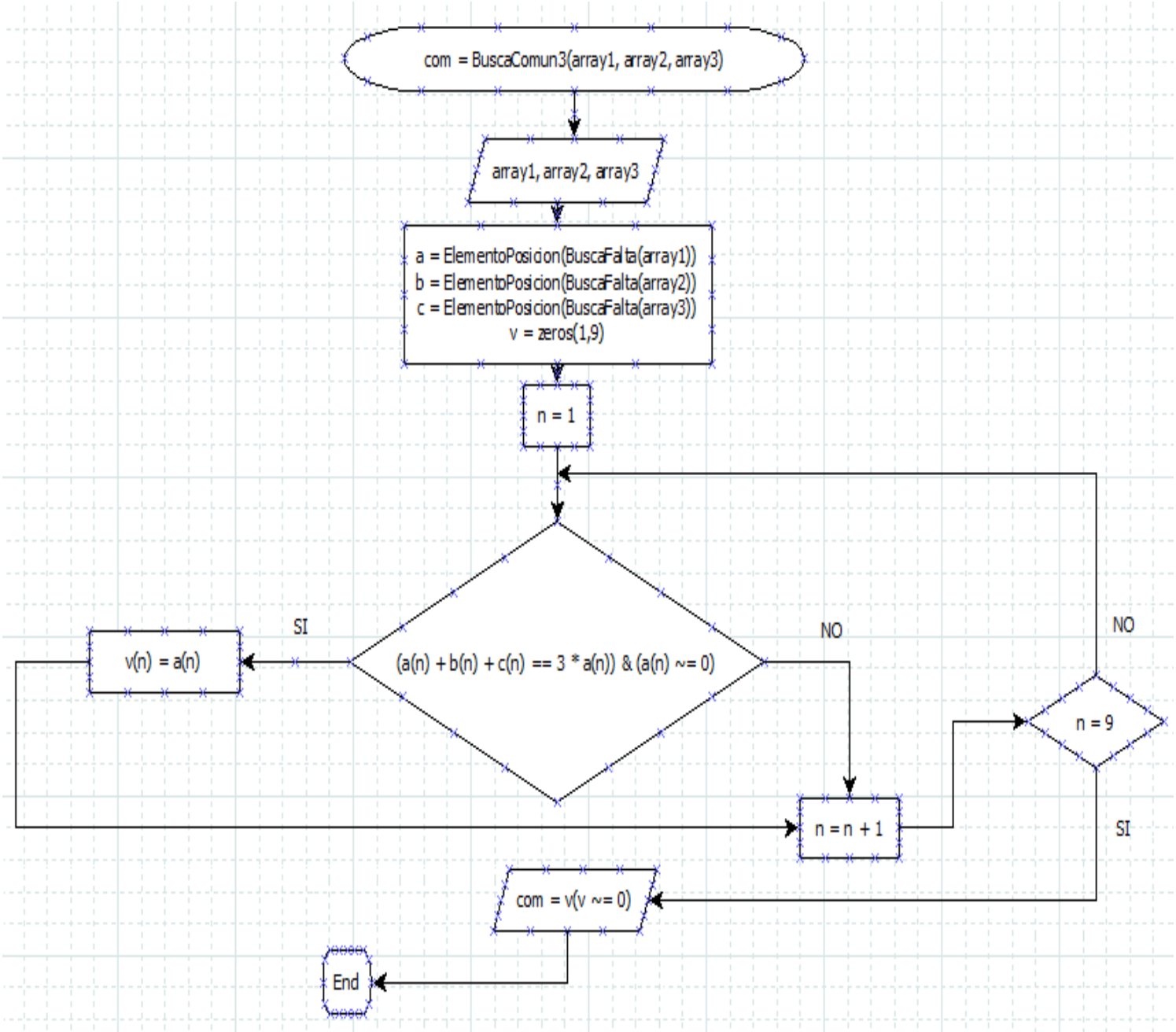
Howmany


num = Howmany(mat,number)
mat, number
[i j] = size(mat)
m = 1
n = 1
vec = mat(mat == number)
z = length(vec)
n = n + 1
n = j
NO
SI
m = m + 1
m = i
NO
SI
num = z
End

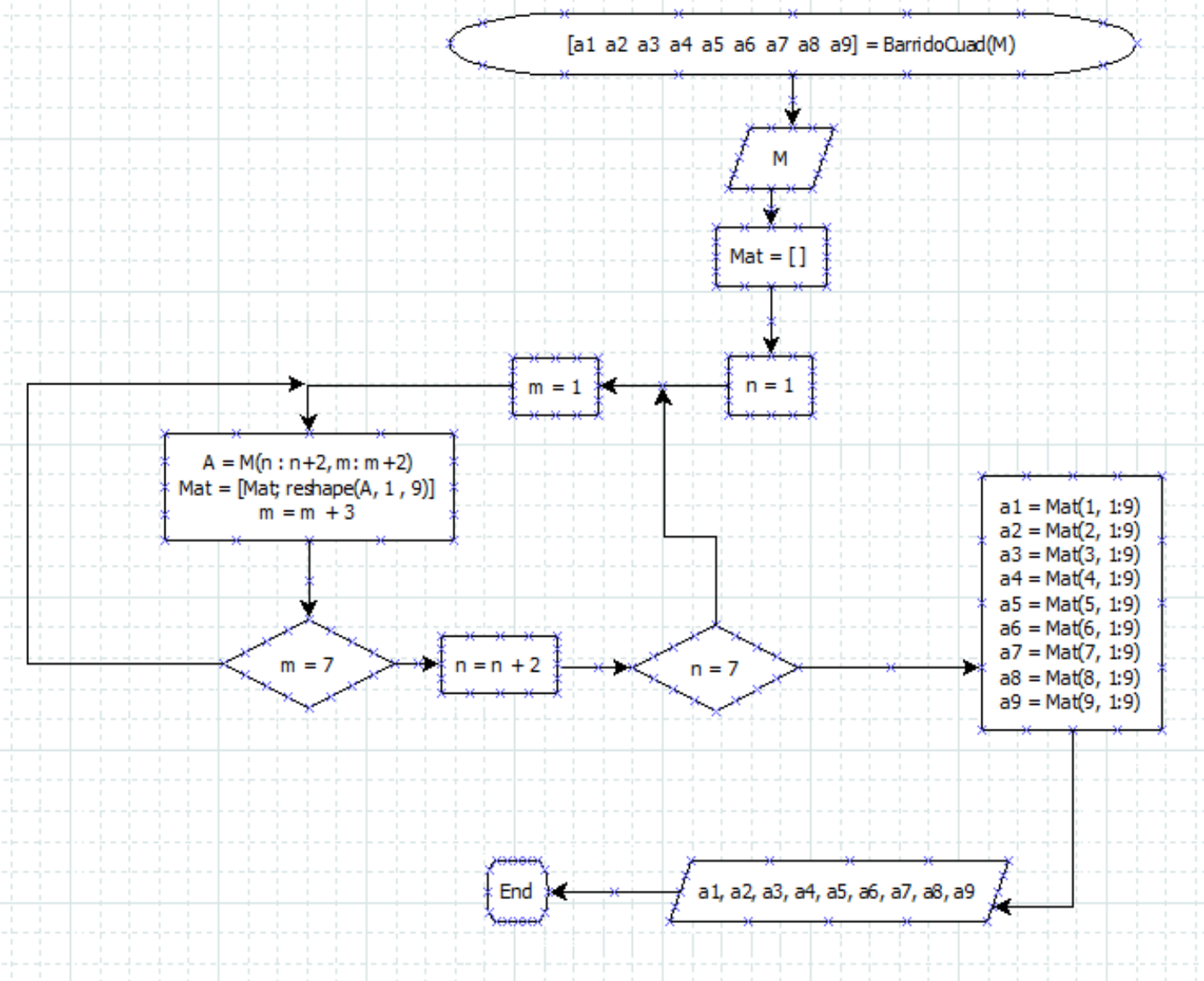
BuscaFalta


vec = BuscaFalta(array)
array
z = ElementoPosicion(array)
vec = 1 : 9
sape = [ ]
m = 1
vec(m) + z(m) == vec(m)
sape = [sape vec(m)]
m = m + 1
m = 9
vec = sape
End

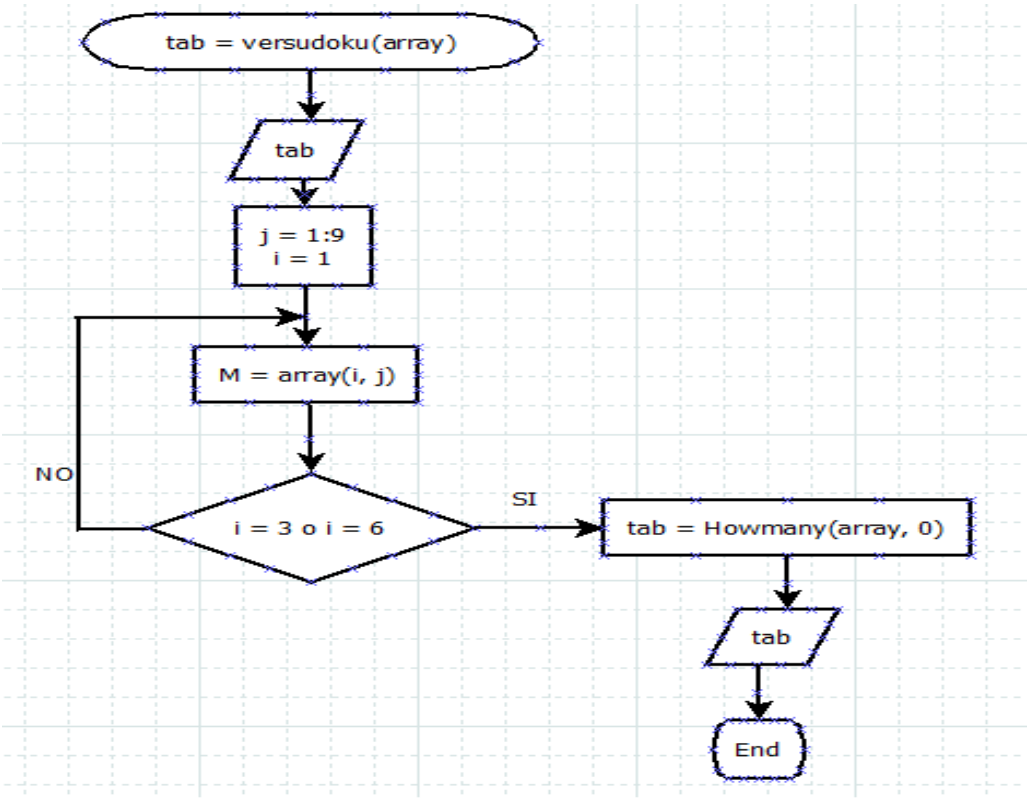
BuscaComun3



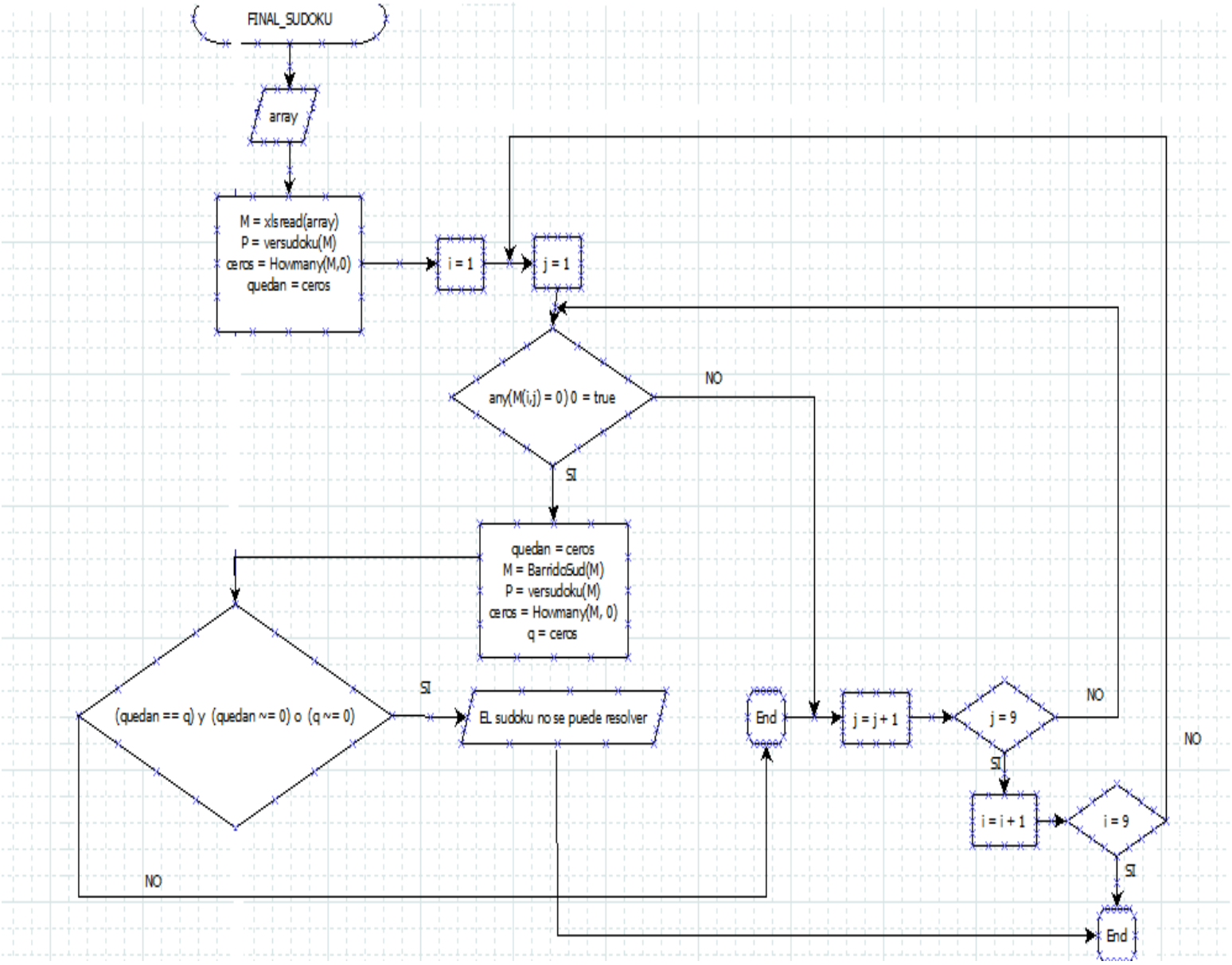
BarridoCuad



VerSudoku


tab = versudoku(array)
tab
j = 1:9
i = 1
M = array(i, j)
NO
i = 3 o i = 6
SI
tab = Howmany(array, 0)
tab
End

FINAL_SUDOKU



FUNCIONES

1. Primarias:

Como hemos visto en los algoritmos, necesitamos crear algunas funciones antes de realizar nuestro script. A partir del análisis para resolver el sudoku, se ha creado dos funciones básicas que apoyan esto. A continuación se presentarán dichas funciones:

a) Elemento Posición:

Esta función evalúa los elementos de un vector de tamaño menor o igual a 9 de modo que los coloca en la posición igual al valor de cada elemento.
Por ejemplo, tenemos el vector:

$$A = [1\ 3\ 7\ 6]$$

Entonces:

$$ElementoPosicion(A) = [1\ 0\ 3\ 0\ 0\ 6\ 7\ 0\ 0]$$

Proceso:

1. Crea un vector de ceros de tamaño 9.
2. Si hay ceros dentro del vector de entrada, se le quitan los ceros.
3. El valor del elemento del vector de entrada es igual a la posición que tomará el mismo valor pero en el vector de ceros.
4. El vector de salida será el nuevo vector de ceros con los valores reemplazados del vector entrada.

Script de la función ElementoPosicion:

```
function z = ElementoPosicion(array)
%
%Funcion 'ElementoPosicion(array)' toma cada elemento de un vector y lo
%         coloca en la posicion igual al valor del elemento.
%
m = zeros(1,9); vec = array(array ~= 0);
for n = 1:length(vec);
    c = vec(n);
    m(c) = vec(n);
end
z = m;
```

b) Howmany:

La función Howmany cuenta el número de veces que se repite un número dentro de una matriz. Para ello nuestros valores de entrada serán el vector definido y el número que quiero saber cuántas veces se repite.
Por ejemplo, tenemos el vector:

$$A = [1\ 4\ 2\ 3\ 1\ 1\ 6\ 8\ 4]$$

Entonces:

$$Howmany(A, 1) = 3$$
$$Howmany(A, 4) = 2$$

Proceso:

1. Se extrae los elementos del vector que sean iguales al valor numérico introducido.
2. El tamaño del nuevo vector será igual a la salida de cuántas veces se repite dicho número en la matriz.

Script de la función Howmany:

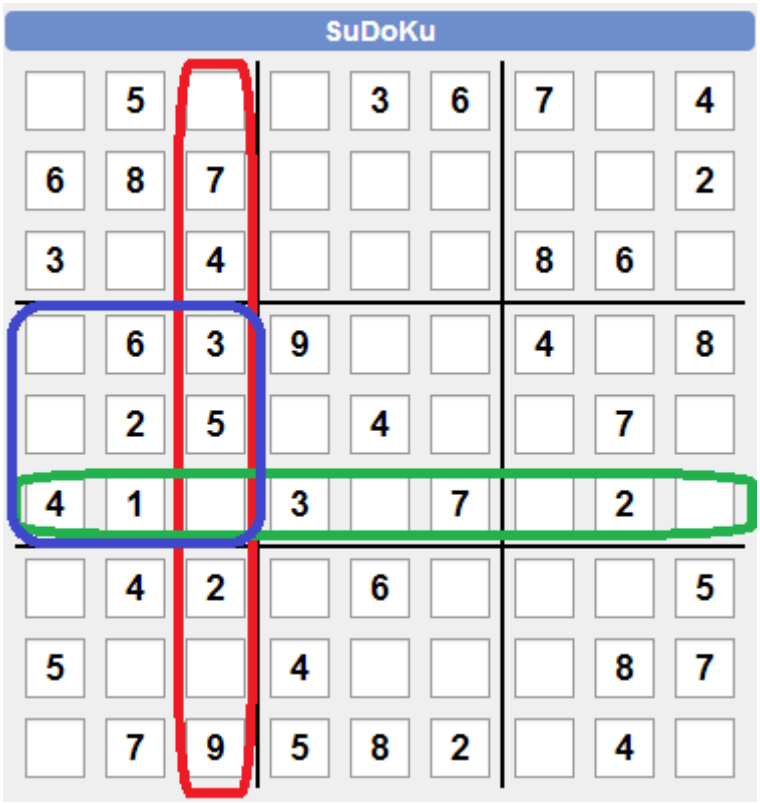
```
function num = Howmany(mat,number)
%
%Funcion 'Howmany' muestra cuantas veces se repite un numero definidos
hay en la matriz.
%
[i j] = size(mat);
for m = 1:i;
    for n = 1:j;
        vec = mat(mat == number);
        z = length(vec);
    end
end
num = z;
```

2. Para el Sudoku:

Luego de que hicimos nuestras funciones básicas, creamos unas especiales para realizar nuestro Sudoku, ayudándonos de las funciones anteriores.

a) BuscaFalta

Esta función busca los números que faltan (del 1 al 9) dentro de un vector (en este caso de la fila, columna o cuadrícula 3x3). La función 'BuscaFalta' nos ayudará para la resolución del sudoku porque si evaluamos para la fila, columna y cuadrícula de 3x3 y encontramos el elemento que se repite en estas 3, ya obtendremos el número que irá en la casilla donde haya un cero.
Por ejemplo, si consideramos el sudoku de la figura 1:



El vector fila está dado por:

$$Fila = [4\ 1\ 0\ 3\ 0\ 7\ 0\ 2\ 0]$$

Entonces, utilizando la función BuscaFalta:

$$Fila = [5\ 6\ 8\ 9]$$

Proceso:

1. Con ayuda de la función ElementoPosicion, ordenamos el vector de entrada.
2. Creamos un vector del 1 al 9.
3. Dentro del vector de entrada habrán elementos de ceros.
4. Por consiguiente, si es que sumamos elemento a elemento de ambos vectores (vector entrada y vector del 1 al 9), y el resultado nos da el valor de uno de los elementos del vector del 1 al 9, ese será el número que falta.

$$Fila = [4\ 1\ 0\ 3\ 0\ 7\ 0\ 2\ 0]$$
$$ElementoPosicion(Fila) = [1\ 2\ 3\ 4\ 0\ 0\ 7\ 0\ 0]$$
$$Vec = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$$
$$ElementoPosicion(Fila) + Vec = [2\ 4\ 6\ 8\ \mathbf{5}\ \mathbf{6}\ 14\ \mathbf{8}\ \mathbf{9}]$$
$$VectorSalida = [5\ 6\ 8\ 9]$$

Script de la función BuscaFalta:

```
function vec = BuscaFalta(array)
%
%Funcion 'vec' genera un vector con los elementos del 1 al 9 que le
faltan
%              al vector analizado ('array').
%
z = ElementoPosicion(array); vec = 1:9;
sape = [ ];
for m = 1:9;
    if vec(m) + z(m) == vec(m);
        sape = [sape vec(m)];
    end
end
vec = sape;
```

b) BuscaComun3

La función 'BuscaComun3' encuentra los elementos comunes entre 3 vectores.
Por ejemplo:

$$A = [1\ 3\ 2\ 7\ 8]$$
$$B = [4\ 2\ 3\ 9]$$
$$C = [2\ 3\ 7\ 8\ 9]$$
$$BuscaComun3(A, B, C) = [2\ 3]$$

Proceso:

1. La función 'BuscaComun3' viene apoyada con las funciones 'ElementoPosicion' y 'BuscaFalta'.
2. Se definen los 3 vectores de entrada, y se hallan los elementos que faltan.
3. Se obtienen 3 vectores con los elementos que faltan en cada vector de entrada.
4. Luego, se suman dichos vectores elemento a elemento y si el resultado es igual al triple de uno de los elementos, entonces ese es el número común de los 3 vectores.

$$A = [4\ 1\ 0\ 3\ 0\ 7\ 0\ 2\ 0]$$
$$BuscaFalta(A) = [5\ 6\ 8\ 9]$$
$$M = ElementoPosicion(BuscaFalta(A)) = [0\ 0\ 0\ 0\ 5\ 6\ 0\ 8\ 9]$$

$$B = [0\ 1\ 0\ 3\ 5\ 0\ 0\ 4\ 0]$$
$$BuscaFalta(B) = [2\ 6\ 7\ 8\ 9]$$
$$N = ElementoPosicion(BuscaFalta(B)) = [0\ 2\ 0\ 0\ 0\ 6\ 7\ 8\ 9]$$
$$C = [2\ 0\ 4\ 5\ 0\ 0\ 7\ 0\ 9]$$
$$BuscaFalta(C) = [1\ 3\ 6\ 8]$$
$$P = ElementoPosicion(BuscaFalta(C)) = [1\ 0\ 3\ 0\ 0\ 6\ 0\ 8\ 0]$$

Sumamos los 3 vectores elemento a elemento:

$$M = [0\ 0\ 0\ 0\ 5\ 6\ 0\ 8\ 9]$$
$$N = [0\ 2\ 0\ 0\ 0\ 6\ 7\ 8\ 9]$$
$$P = [1\ 0\ 3\ 0\ 0\ 6\ 0\ 8\ 0]$$

$$M + N + P = [1\ 2\ 3\ 0\ 5\ \mathbf{18}\ 7\ \mathbf{24}\ 18]$$

$$BuscaComun3(A, B, C) = \left[\frac{18}{3}, \frac{24}{3}\right]$$

$$BuscaComun3(A, B, C) = [6\ 8]$$

Script de la función BuscaComun3:

```
function com = BuscaComun3(array1,array2,array3)
%
%
%
a = ElementoPosicion(BuscaFalta(array1));
b = ElementoPosicion(BuscaFalta(array2));
c = ElementoPosicion(BuscaFalta(array3));
v = zeros(1,9);
for n = 1:9;
    if (a(n) + b(n) + c(n)) == 3*(a(n)) & (a(n) ~= 0);
        v(n) = a(n);
    end
end
com = v(v ~= 0);
```

c) BarridoCuad

Esta función extrae las matrices o cuadrículas de 3x3 del sudoku de 9x9; por lo tanto, tendremos 9 matrices y las convierte a vectores.

Script de la función BarridoCuad:

```
function [a1 a2 a3 a4 a5 a6 a7 a8 a9] = BarridoCuad(M)
%
%
%
Mat = [ ];
for n = 1:3:7;
    for m = 1:3:7;
        A = M(n:n+2,m:m+2);
        Mat = [Mat;reshape(A,1,9)];
    end
end
a1 = Mat(1,1:9);a2 = Mat(2,1:9);a3 = Mat(3,1:9);
a4 = Mat(4,1:9);a5 = Mat(5,1:9);a6 = Mat(6,1:9);
a7 = Mat(7,1:9);a8 = Mat(8,1:9);a9 = Mat(9,1:9);
```

d) BarridoSud

Esta función hace el barrido para cada elemento cero que esté dentro del tablero del sudoku de 9x9.

Proceso:

1. A partir de la función BarridoCuad, se obtendrán 9 vectores que salieron de las cuadrículas de 3x3.
2. Se verificaran los elementos del tablero, si un elemento es diferente de cero, se continuará a la siguiente casilla.
3. Si algún elemento del tablero es cero, el análisis comenzará de la siguiente manera:
   - Se extraen los vectores filas, columna y cuadrícula 3x3.
   - Con la función BuscaFalta y BuscaComun3, verificamos el número de elementos faltantes que se repiten.
   - Si solo hay un elemento faltante que se repite en los 3 vectores, entonces ese es el valor que ocupa dicha casilla. De lo contrario se seguirá analizando las demás.
4. El análisis terminará cuando no quede ningún cero en el sudoku.

Script de la función BarridoSud:

```
function z = BarridoSud(M)
 [a1 a2 a3 a4 a5 a6 a7 a8 a9] = BarridoCuad(M);
for i = 1:9;
    for j = 1:9;
        if M(i,j) == 0;
            vec1 = ElementoPosicion(M(i,1:9));
            vec2 = ElementoPosicion((M(1:9,j))');
            if (j>=1 & j<=3) & (i>=1 & i<=3)
                w = BuscaComun3(vec1,vec2,a1);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=4 & j<=6) & (i>=1 & i<=3)
                w = BuscaComun3(vec1,vec2,a2);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=7 & j<=9) & (i>=1 & i<=3)
                w = BuscaComun3(vec1,vec2,a3);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=1 & j<=3) & (i>=4 & i<=6)
                w = BuscaComun3(vec1,vec2,a4);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=4 & j<=6) & (i>=4 & i<=6)
```

```
                w = BuscaComun3(vec1,vec2,a5);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=7 & j<=9) & (i>=4 & i<=6)
                w = BuscaComun3(vec1,vec2,a6);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=1 & j<=3) & (i>=7 & i<=9)
                w = BuscaComun3(vec1,vec2,a7);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=4 & j<=6) & (i>=7 & i<=9)
                w = BuscaComun3(vec1,vec2,a8);
                if length(w) == 1
                    M(i,j) = w;
                end
            elseif (j>=7 & j<=9) & (i>=7 & i<=9)
                w = BuscaComun3(vec1,vec2,a9);
                if length(w) == 1
                    M(i,j) = w;
                end
            end
        end
    end
end
z = M;
```

e) VerSudoku

Esta función genera una tabla de sudoku común para la visualización en la Ventana de Comandos de Matlab. Utilizaremos columnas y filas de separación a base de los símbolos: << | >> y << - - >>, respectivamente.

Proceso:

1. Se produce el título a visualizar (‘JUEGO DEL SUDOKU’).
2. Se genera un vector general que dará la forma de la tabla del sudoku. Por ejemplo:

$$[D1 \quad D2 \quad D3 \quad | \quad D4 \quad D5 \quad D6 \quad | \quad D7 \quad D8 \quad D9]$$

3. Este vector se repetirá para las 8 filas siguientes, por lo que ya tendremos nuestras columnas de separación en la tabla del sudoku.
4. Ahora bien, falta hacer las filas de separación, las cuales haremos de la siguiente forma:

$$[D1 \quad D2 \quad D3 \quad | \quad D4 \quad D5 \quad D6 \quad | \quad D7 \quad D8 \quad D9]$$
$$[- -]$$

5. Se necesita generar estas filas y columnas cada 3 elementos de la matriz; de esta forma, se verá como una tabla de sudoku común.

6. Como el vector general está dado por filas, haremos que para cada fila dentro de nuestra matriz se muestre el vector general ya mencionado.
7. Sin embargo, necesitamos que haya también líneas de separación horizontal las cuales serán cada 3 filas. Así que en el momento del proceso cuando el muestreo llegue a la fila 3 o fila 6 se deberá mostrar una línea de separación horizontal.
8. Finalmente, si introducimos la matriz de 9x9, la función nos debe mostrar la misma dentro de una tabla con filas y columnas de separación.

Script de la función VerSudoku:

```
function tab = versudoku(array)
%
%
%
j = 1:9;
fprintf('\t\t\t\t\t\tJUEGO DEL SUDOKU\n');
fprintf('\t\t\t\t\t\t================\n');
disp(' ');
for i = 1:9;
    M = array(i,j);
    fprintf(['\t\t\t%.0f\t%.0f\t%.0f\t' '|' '\t%.0f\t%.0f\t%.0f\t' '|'
'\t%.0f\t%.0f\t%.0f\n'],M)
    if i == 3 | i == 6;
        fprintf(['\t\t\t------------' '|' '---------------' '|' '--------
----\n'])
    end
end
tab = Howmany(array,0);
```

PROGRAMA_SUDOKU

Ahora ya estamos listos para generar el programa que dirigirá todas las funciones para la resolución de cualquier sudoku en modo fácil o normal. En este caso, estamos exceptuando los del modo difícil porque en un primer barrido nunca se encontrará un solo número faltante que se repita en la fila, columna y cuadrícula de 3x3; por lo tanto, no se podrá resolver por el método que estamos empleando.

A partir del diagrama de flujo que hicimos anteriormente para la solución de nuestro sudoku, obtenemos una más simplificada con ayuda de nuestras funciones ya creadas. Además, debemos tener en cuenta que para obtener nuestra matriz de 9x9 donde simularemos el sudoku de modo fácil o normal, la haremos en el software Microsoft Excel, desde donde lo exportaremos hacia Matlab. Para ello usaremos el comando 'xlsread('nombre_archivo') para exportarlo a nuestra ventana de comandos.

A continuación, presentamos el script que se creó para el desarrollo del programa FINAL_SUDOKU:

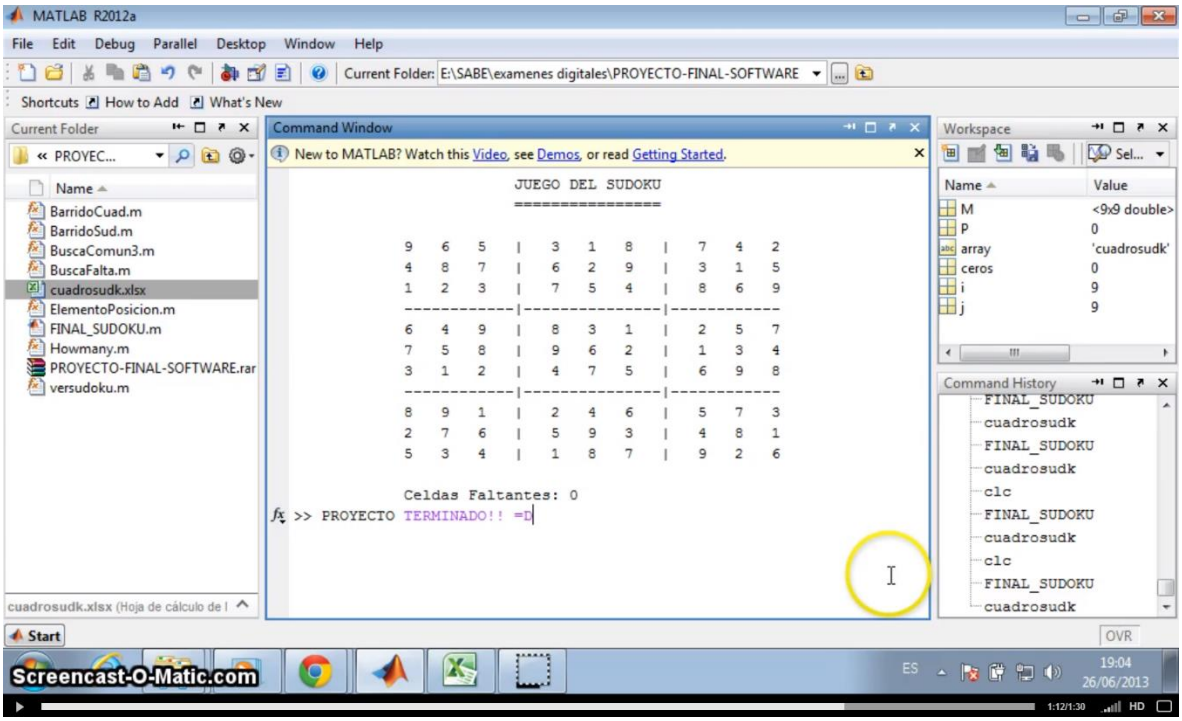
```
clear all
close all
clc
%%%%%%%%%%%%%%%%%%

array = input('Ingrese nombre del archivo excel: ','s');
clc
M = xlsread(array);
P = versudoku(M);
ceros = Howmany(M,0);
quedan = ceros;
disp(' ');
fprintf('\t\t\tCeros Restantes: %.0i\n',ceros);
pause(0.5)
for i = 1:9;
    for j = 1:9;
        while any(M(i,j) == 0) == true
            clc
            quedan = ceros;
            M = BarridoSud(M);
            P = versudoku(M);
            ceros = Howmany(M,0);
            q = ceros;
            disp(' ');
            fprintf('\t\t\tCeldas Faltantes: %.0f\n',ceros);
            if (quedan == q) & (quedan ~= 0) & (q ~= 0)
                disp(' ');
                fprintf('\t\t\tEl sudoku no se puede resolver\n\t\t\tDale
manito arriba si te gustó.\n')
                break
            end
           pause(0.5)
        end
    end
end
```

PRUEBAS Y RESULTADOS

Durante las semanas 14 y 15 se hicieron muchas pruebas con diferentes scripts, se tuvieron que modificar algunas funciones por la extensión que tenían las mismas. Una función tiene que ser corta y ayudar a simplificar el script general donde se trabajará el proyecto.

Dentro del proyecto se creó un contador de celdas, el cual sirve para indicar cuántas celdas faltan por rellenar. Además, al exportar el archivo Excel a Matlab se generó una tabla donde se coloquen todos los datos, de esta forma tomaría la mejor forma que se asemeje a

un sudoku. Se colocaron pausas de 0.5 segundos para que el programa vaya mostrando como se va resolviendo el sudoku en la ventana de comandos.
Cabe resaltar que en nuestro sudoku no tomamos en cuenta el hecho de que donde están los ceros debían ir espacios en blancos, pues para hacerlo se necesita de un proceso más complejo. Esto se debe a que al exportarlo de Excel hacia Matlab, este último lo reconoce como un NAN y lo que se quiere son espacios en blancos que para nuestro proyecto no ha sido evaluado.
Al finalizar, se obtuvo un buen proyecto que hace que funcione la resolución del sudoku.



Enlace:
(http://www.facebook.com/photo.php?v=3246197129543&set=vb.1705372136&type=2&theater)

BIBLIOGRAFIA

- SUDOKU-ONLINE [Página web sobre sudokus y mahjong] (2012) (http://www.sudoku-online.org/)
- GILAT, Amos. Matlab: Una introducción con ejemplos prácticos. 2º Ed. Barcelona – España: Reverté S.A. (2006)

View publication stats