%Validation\_nQueens\_Problem\_Solution

%{

This program is intended:

a) *to verify the correctness of the solution n-Queens Problem for an arbitrary chessboard of size n x n,*

b) *to verify the correctness of an arbitrary composition of k queens randomly distributed on a chessboard of size n x n.*

%}

%{

License: *Attribution-NonCommercial-ShareAlike CC BY-NC-SA*

“This license lets others remix, adapt, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms”.

%}

%{

Project author and developer - Grigoryan Eros (EricGrig), 2020

I will be glad if any sections of the code, or the entire program as a whole, will be used for scientific purposes, or for education. At the same time, I will be grateful if you consider it possible to refer to my publication. It is an element of culture and a sign of mutual respect.

For commercial use of any part of the program code, or the entire program as a whole, the written consent of the author is required.

%}

%{

This algorithm is described in detail in the article: *Grigoryan E., Linear algorithm for solution n-Queens Completion problem,* [*https://arxiv.org/abs/1912.05935*](https://arxiv.org/abs/1912.05935) *.* It will be correct if you first read this publication before begin to analyse the source code. This will make the program description more transparent and reduce the number of possible questions.

%}

%{

*How is prepared the initial data*?

Denote the chessboard side size by *n*.

Let there be a one-dimensional nullified array of size *n*.

If in the *i-th* row of the chessboard the queen is placed in position *j*, then, respectively, in the *i-th* cell of the data array the value of *j* is written.

%}

%{

*Data input*.

We will write the input data to the *Q* array.

The name of the input file *nQueens\_Test\_Solution.mat* should be changed, if another file is selected

%}

inputFileName= 'nQueens\_Test\_Completion\_Solution.mat';

%inputFileName= 'kQueens\_Test\_Composition.mat';

iInfo=['Input file name: ' inputFileName];

disp(iInfo);

Q=importdata(inputFileName);

% Display the size of a chessboard

n=length(Q);

disp(' ');

tStr = sprintf(' The size of a chessboard = %d',n);

disp(tStr);

disp(' ')

% *n2* – the size of diagonal control arrays

n2=n\*2;

% Formation of control arrays for diagonal control.

D1=zeros(1,n2,'uint8');

D2=zeros(1,n2,'uint8');

B=zeros(1,n,'uint8');

%{

We determine the number of rows on the chessboard in which the queen is not located, i.e. number of free rows. This number is equal to the total number of zero cells in the *Q* array.

%}

nZero=sum(Q==0);

%{ *qError* – decision validation index. If *qError == 0*, then the solution is correct, otherwise, if *qError == 1*, then the solution contains an error.

%}

tic

%{

If *nZero == 0*, then this means that input data is a *solution of the n-Queens Problem*, otherwise - the input data is *a composition*.

%}

%{

The essence of the algorithm is that we can only once write *1* in zeroed cells of the control arrays *D1, D2*, and *B*, otherwise, one of the three conditions of the task will be violated.

%}

if nZero==0

for i=1:n

j=Q(i);

r=n+j-i;

t=j+i;

if D1(r)==0 && D2(t)==0 && B(j)==0

D1(r)=1;

D2(t)=1;

B(j)=1;

else

qError=1;

break

end

end

else

for i=1:n

if Q(i)>0

j=Q(i);

r=n+j-i;

t=j+i;

if D1(r)==0 && D2(t)==0 && B(j)==0

D1(r)=1;

D2(t)=1;

B(j)=1;

else

qError=1;

break

end

end

end

end

toc

% *k* –the number of nonzero elements in the array *Q* (composition size)

k=n-nZero;

% The output the results of analysis

if qError==0

if nZero==0

disp('Solutions is ok!')

else

tStr = sprintf('Composition size = %d',k);

disp(tStr);

disp('Composition is ok!')

end

else

tStr = sprintf('Error in solution in row = %d',i);

disp(tStr)

end