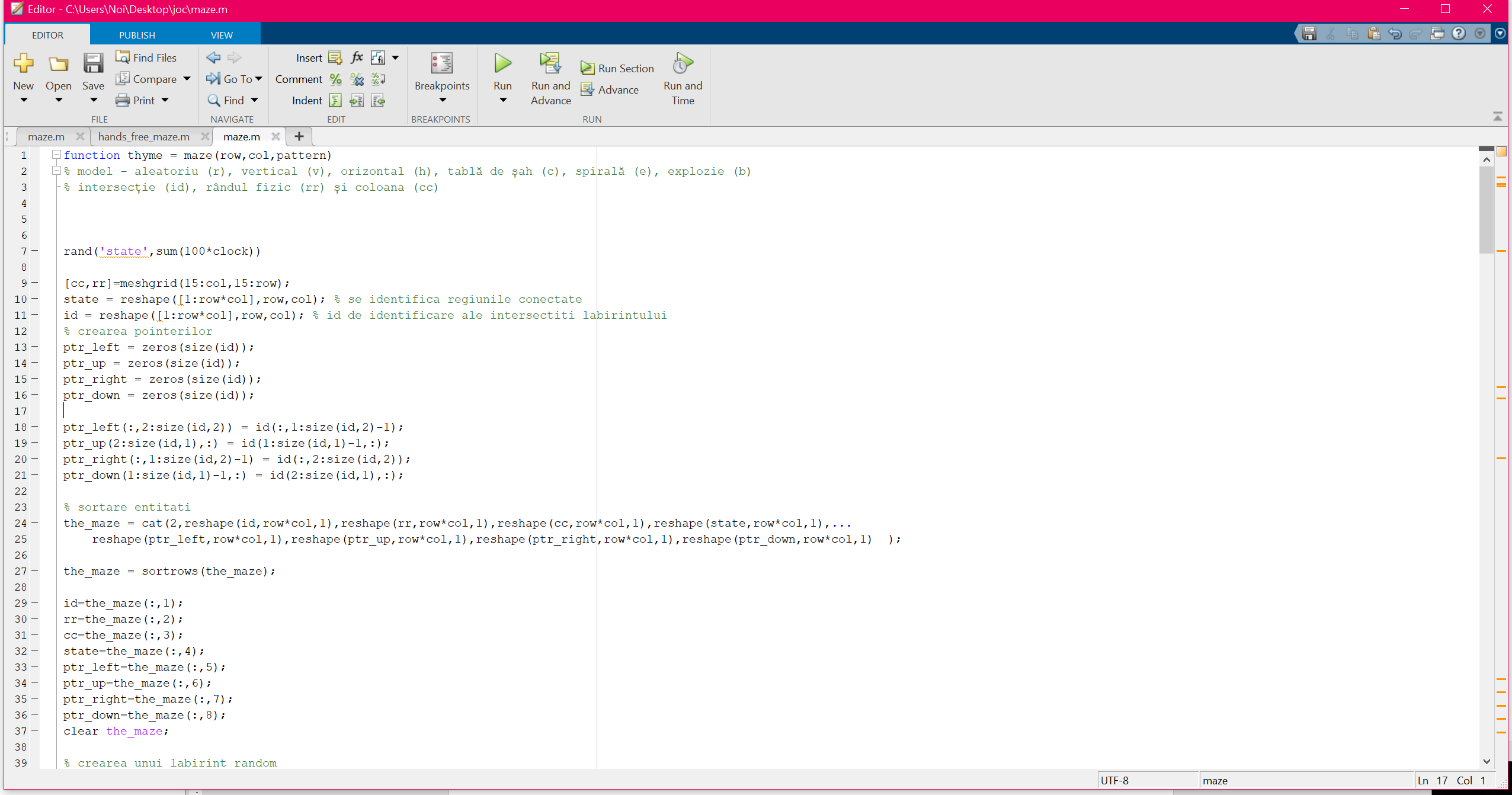
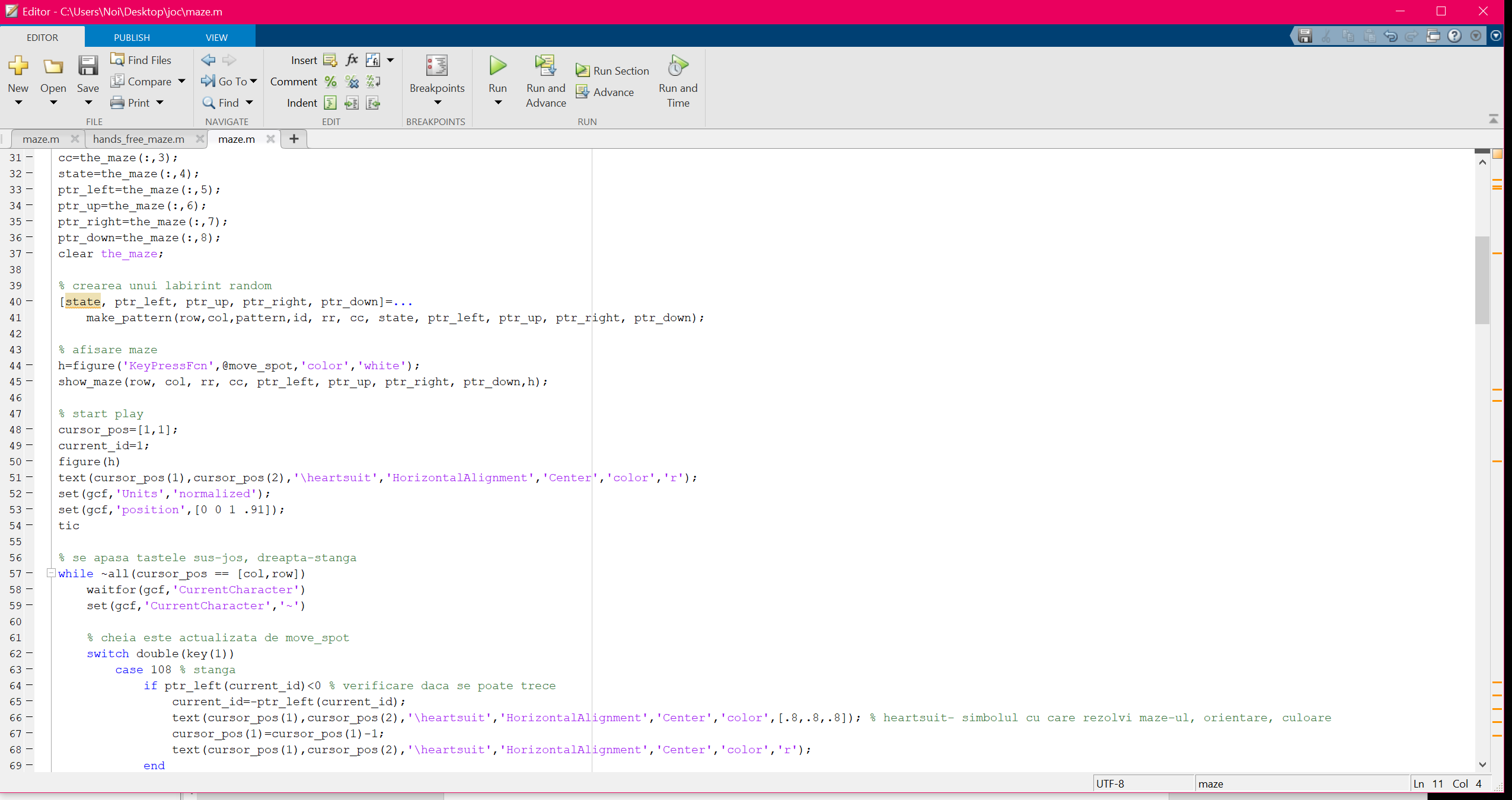
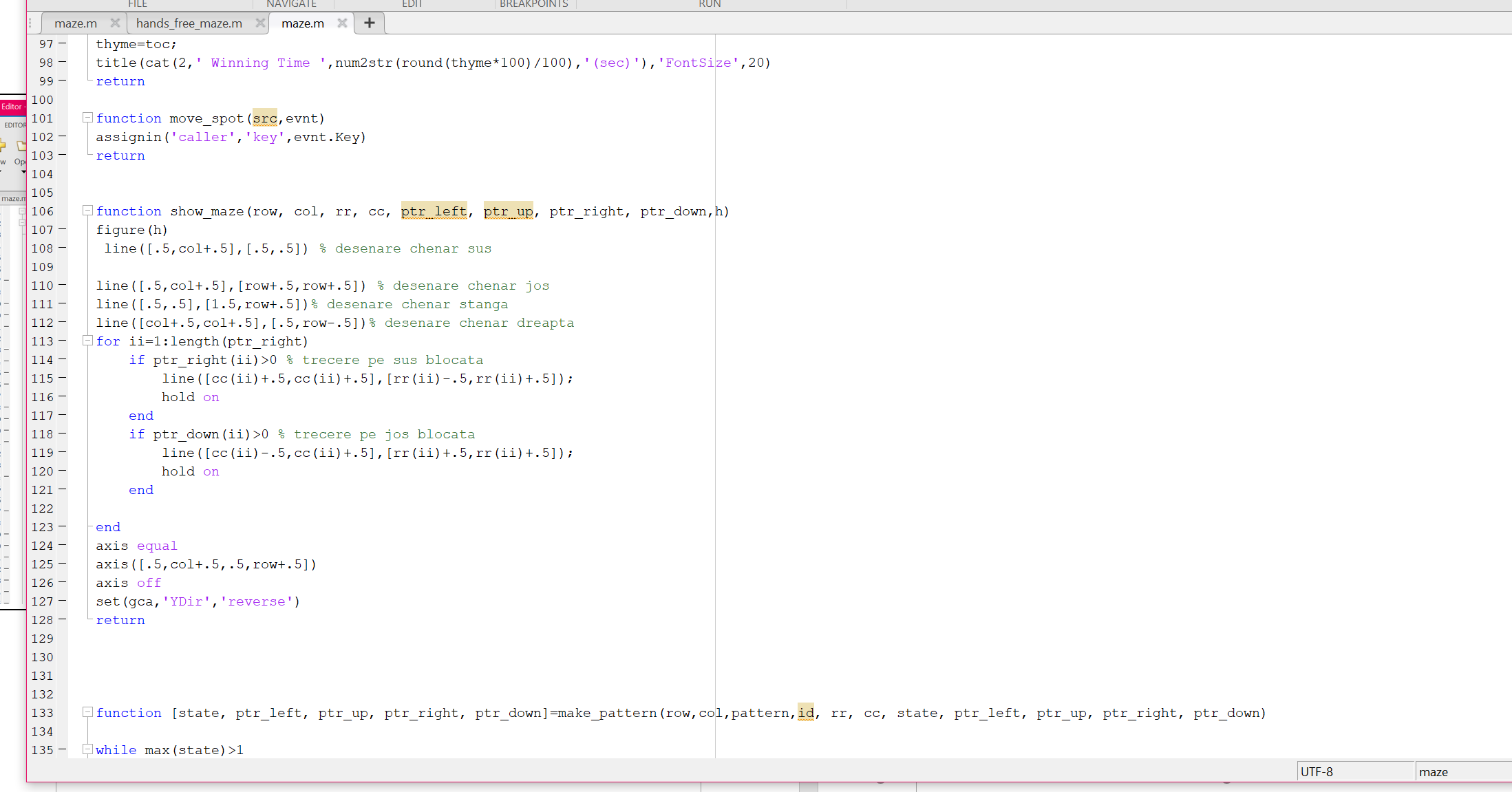
MAZE

Cerinta: 15 Realizați o interfață grafică, utilizând mediul Maltab, pentru crearea unui joc functional de tipul Labirint (ex. butoane cu diferite functionalitati: inainte, inapoi, stanga, dreapta, nivel, scor  
etc.)

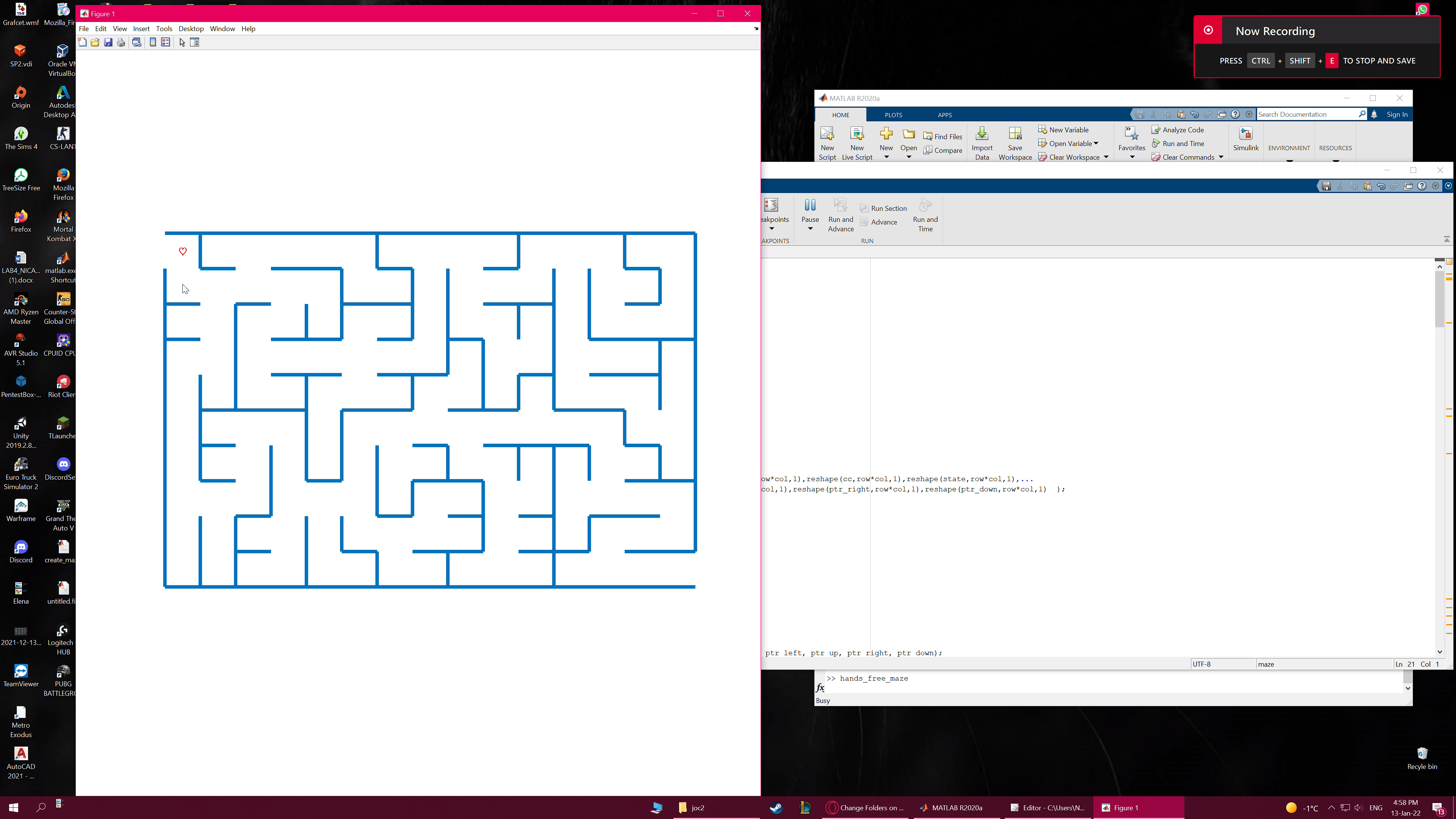
Cod:

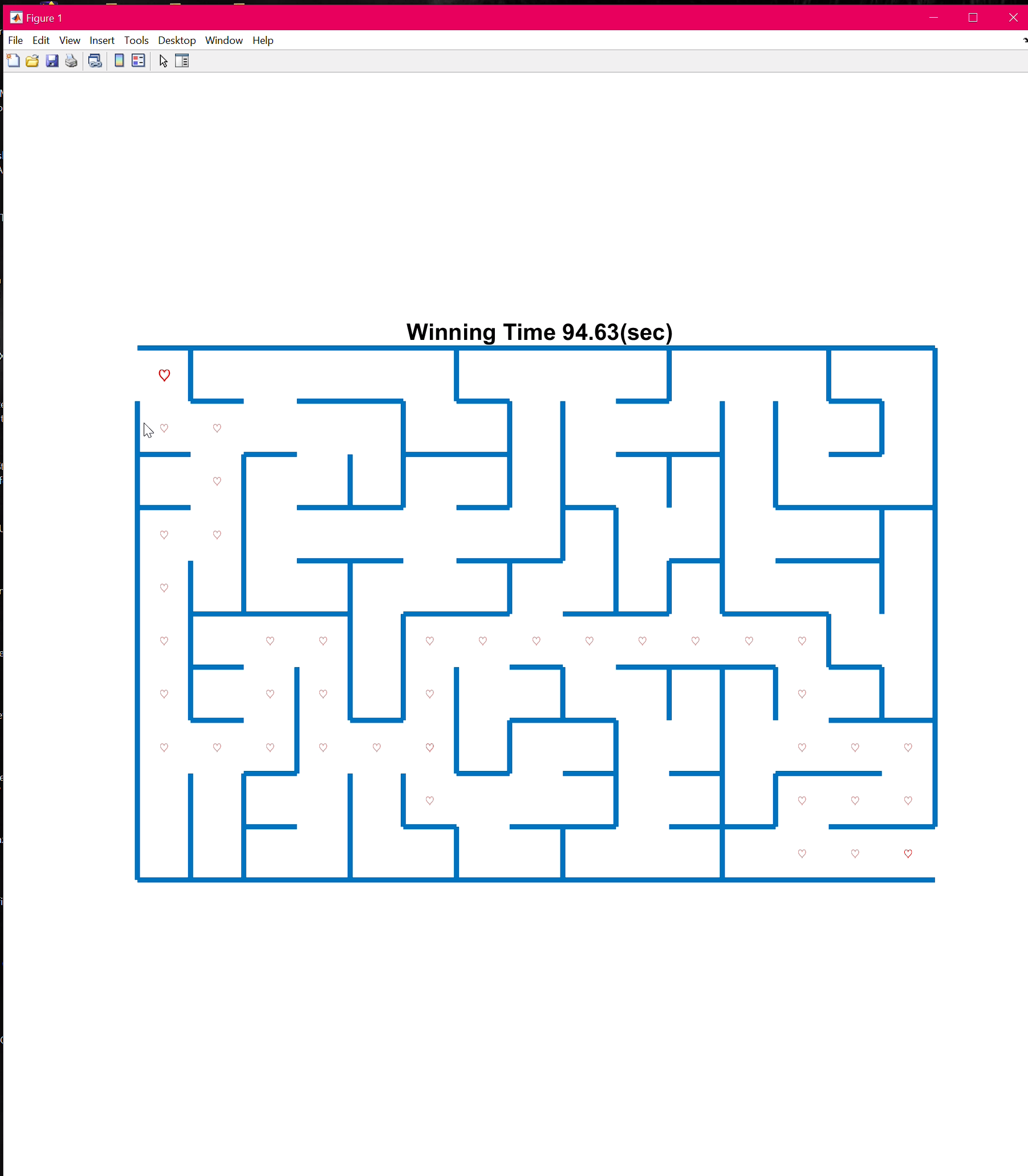






Screen-uri labirint:





Cod sursa:

function thyme = maze(row,col,pattern)

% model - aleatoriu (r), vertical (v), orizontal (h), tabl? de ?ah (c), spiral? (e), explozie (b)

% intersec?ie (id), rândul fizic (rr) ?i coloana (cc)

rand('state',sum(100\*clock))

[cc,rr]=meshgrid(15:col,15:row);

state = reshape([1:row\*col],row,col); % se identifica regiunile conectate

id = reshape([1:row\*col],row,col); % id de identificare ale intersectiti labirintului

% crearea pointerilor

ptr\_left = zeros(size(id));

ptr\_up = zeros(size(id));

ptr\_right = zeros(size(id));

ptr\_down = zeros(size(id));

ptr\_left(:,2:size(id,2)) = id(:,1:size(id,2)-1);

ptr\_up(2:size(id,1),:) = id(1:size(id,1)-1,:);

ptr\_right(:,1:size(id,2)-1) = id(:,2:size(id,2));

ptr\_down(1:size(id,1)-1,:) = id(2:size(id,1),:);

% sortare entitati

the\_maze = cat(2,reshape(id,row\*col,1),reshape(rr,row\*col,1),reshape(cc,row\*col,1),reshape(state,row\*col,1),...

reshape(ptr\_left,row\*col,1),reshape(ptr\_up,row\*col,1),reshape(ptr\_right,row\*col,1),reshape(ptr\_down,row\*col,1) );

the\_maze = sortrows(the\_maze);

id=the\_maze(:,1);

rr=the\_maze(:,2);

cc=the\_maze(:,3);

state=the\_maze(:,4);

ptr\_left=the\_maze(:,5);

ptr\_up=the\_maze(:,6);

ptr\_right=the\_maze(:,7);

ptr\_down=the\_maze(:,8);

clear the\_maze;

% crearea unui labirint random

[state, ptr\_left, ptr\_up, ptr\_right, ptr\_down]=...

make\_pattern(row,col,pattern,id, rr, cc, state, ptr\_left, ptr\_up, ptr\_right, ptr\_down);

% afisare maze

h=figure('KeyPressFcn',@move\_spot,'color','white');

show\_maze(row, col, rr, cc, ptr\_left, ptr\_up, ptr\_right, ptr\_down,h);

% start play

cursor\_pos=[1,1];

current\_id=1;

figure(h)

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color','r');

set(gcf,'Units','normalized');

set(gcf,'position',[0 0 1 .91]);

tic

% se apasa tastele sus-jos, dreapta-stanga

while ~all(cursor\_pos == [col,row])

waitfor(gcf,'CurrentCharacter')

set(gcf,'CurrentCharacter','~')

% cheia este actualizata de move\_spot

switch double(key(1))

case 108 % stanga

if ptr\_left(current\_id)<0 % verificare daca se poate trece

current\_id=-ptr\_left(current\_id);

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color',[.8,.8,.8]); % heartsuit- simbolul cu care rezolvi maze-ul, orientare, culoare

cursor\_pos(1)=cursor\_pos(1)-1;

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color','r');

end

case 114 % dreapta

if ptr\_right(current\_id)<0 % verificare daca se poate trece

current\_id=-ptr\_right(current\_id);

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color',[.8,.8,.8]);

cursor\_pos(1)=cursor\_pos(1)+1;

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color','r');

end

case 117 % sus

if ptr\_up(current\_id)<0 % verificare daca se poate trece

current\_id=-ptr\_up(current\_id);

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color',[.8,.8,.8]);

cursor\_pos(2)=cursor\_pos(2)-1;

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color','r');

end

case 100 % jos

if ptr\_down(current\_id)<0 % verificare daca se poate trece

current\_id=-ptr\_down(current\_id);

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color',[.8,.8,.8]);

cursor\_pos(2)=cursor\_pos(2)+1;

text(cursor\_pos(1),cursor\_pos(2),'\heartsuit','HorizontalAlignment','Center','color','r');

end

otherwise

end

end

thyme=toc;

title(cat(2,' Winning Time ',num2str(round(thyme\*100)/100),'(sec)'),'FontSize',20)

return

function move\_spot(src,evnt)

assignin('caller','key',evnt.Key)

return

function show\_maze(row, col, rr, cc, ptr\_left, ptr\_up, ptr\_right, ptr\_down,h)

figure(h)

line([.5,col+.5],[.5,.5]) % desenare chenar sus

line([.5,col+.5],[row+.5,row+.5]) % desenare chenar jos

line([.5,.5],[1.5,row+.5])% desenare chenar stanga

line([col+.5,col+.5],[.5,row-.5])% desenare chenar dreapta

for ii=1:length(ptr\_right)

if ptr\_right(ii)>0 % trecere pe sus blocata

line([cc(ii)+.5,cc(ii)+.5],[rr(ii)-.5,rr(ii)+.5]);

hold on

end

if ptr\_down(ii)>0 % trecere pe jos blocata

line([cc(ii)-.5,cc(ii)+.5],[rr(ii)+.5,rr(ii)+.5]);

hold on

end

end

axis equal

axis([.5,col+.5,.5,row+.5])

axis off

set(gca,'YDir','reverse')

return

function [state, ptr\_left, ptr\_up, ptr\_right, ptr\_down]=make\_pattern(row,col,pattern,id, rr, cc, state, ptr\_left, ptr\_up, ptr\_right, ptr\_down)

while max(state)>1

tid=ceil(col\*row\*rand(15,1));

cityblock=cc(tid)+rr(tid); % get distanta de la start

is\_linked=(state(tid)==1); % Starea de pornire este în regiunea 1

temp = sortrows(cat(2,tid,cityblock,is\_linked),[3,2]); % sortare id uri

tid = temp(1,1);

switch upper(pattern)

case 'C' % principiul tabla de sah

dir = ceil(8\*rand);

nb=3;

block\_size = min([row,col])/nb;

while block\_size>12

nb=nb+2;

block\_size = min([row,col])/nb;

end

odd\_even = (ceil(rr(tid)/block\_size)\*ceil(col/block\_size) + ceil(cc(tid)/block\_size));

if odd\_even/2 == floor(odd\_even/2)

if dir>6

dir=4;

end

if dir>4

dir=3;

end

else

if dir>6

dir=2;

end

if dir>4

dir=1;

end

end

case 'B' % burst

dir = ceil(8\*rand);

if abs((rr(tid)-row/2))<abs((cc(tid)-col/2))

if dir>6

dir=4;

end

if dir>4

dir=3;

end

else

if dir>6

dir=2;

end

if dir>4

dir=1;

end

end

case 'S' % spirala

dir = ceil(8\*rand);

if abs((rr(tid)-row/2))>abs((cc(tid)-col/2))

if dir>6

dir=4;

end

if dir>4

dir=3;

end

else

if dir>6

dir=2;

end

if dir>4

dir=1;

end

end

case 'V'

dir = ceil(8\*rand);

if dir>6

dir=4;

end

if dir>4

dir=3;

end

case 'H'

dir = ceil(8\*rand);

if dir>6

dir=2;

end

if dir>4

dir=1;

end

otherwise % aleatoriu

dir = ceil(4\*rand);

end

% dup? ce este g?sit o forma pentru îndep?rtarea peretelui, forma trebuie s? promoveze

% doua conditii. 1) nu este un perete exterior 2) regiunile pe fiecare parte a peretelui a fost anterior neconectat?.

% Dac? are succes perete este eliminat, st?rile conectate sunt actualizate la cea mai mic? dintre

% cele dou? st?ri, pointerii dintre intersec?iile conectate sunt

% acum negativ.

switch dir

case -1

case 1

if ptr\_left(tid)>0 & state(tid)~=state(ptr\_left(tid))

state( state==state(tid) | state==state(ptr\_left(tid)) )=min([state(tid),state(ptr\_left(tid))]);

ptr\_right(ptr\_left(tid))=-ptr\_right(ptr\_left(tid));

ptr\_left(tid)=-ptr\_left(tid);

end

case 2

if ptr\_right(tid)>0 & state(tid)~=state(ptr\_right(tid))

state( state==state(tid) | state==state(ptr\_right(tid)) )=min([state(tid),state(ptr\_right(tid))]);

ptr\_left(ptr\_right(tid))=-ptr\_left(ptr\_right(tid));

ptr\_right(tid)=-ptr\_right(tid);

end

case 3

if ptr\_up(tid)>0 & state(tid)~=state(ptr\_up(tid))

state( state==state(tid) | state==state(ptr\_up(tid)) )=min([state(tid),state(ptr\_up(tid))]);

ptr\_down(ptr\_up(tid))=-ptr\_down(ptr\_up(tid));

ptr\_up(tid)=-ptr\_up(tid);

end

case 4

if ptr\_down(tid)>0 & state(tid)~=state(ptr\_down(tid))

state( state==state(tid) | state==state(ptr\_down(tid)) )=min([state(tid),state(ptr\_down(tid))]);

ptr\_up(ptr\_down(tid))=-ptr\_up(ptr\_down(tid));

ptr\_down(tid)=-ptr\_down(tid);

end

otherwise

dir

error('quit')

end

end

return