```
%================================
%          ESCAPE PANIC
%       including flooding
%================================
% Marcel Thielmann & Fabio Crameri

function [AGENT] = EscapePanic(Parameter,BuildingList,
   ExitList,StartingList,Plotting)

% if nargin == 0
%     clear;
%
   ************************************************
%     Marcels:
%     TwoExitsStandardSetup;
%
   ************************************************
%     Fabios:
%     [Parameter,BuildingList,ExitList,StartingList,
   Plotting] = SetupModel;
%
   ************************************************
% end


% workflow control
PlotSetup       = false;
PlotEvolution   = Plotting.PlotEvolution;

DirectExitPath  = Parameter.DirectExitPath;
WithAgents      = Parameter.WithAgents;
WithTopo        = Parameter.WithTopo;
WithFlood       = Parameter.WithFlood;
Z_flood         = Parameter.z0_flood;    dzdt_flood =
   Parameter.dzdt_flood;
dangerousDepth  = Parameter.dangerousDepth;

%=========================================================================

% add necessary paths
%=========================================================================

addpath ../DecisionStrategy/
addpath ../WallForces/
addpath ../Plotting/
addpath ../kdtree_alg_OSX/
```

```
addpath ../FastMarching_version3b, add_function_paths
    ();

%=========================================================================

% initialize grid
%=========================================================================

resolution          = Parameter.resolution;
xmin                = Parameter.xmin;
xmax                = Parameter.xmax;
ymin                = Parameter.ymin;
ymax                = Parameter.ymax;

xvec                = xmin:resolution:xmax;
yvec                = ymin:resolution:ymax;
[X_Grid,Y_Grid]     = meshgrid(xvec,yvec);

% set topography
if (strcmp(Parameter.Topo_name,'none') && WithTopo)
    A = 2;
    x0 = 10;
    y0 = 5;
    sigma_x = 15;
    sigma_y = 6;
    Z_Grid = A.*exp(-1*( (X_Grid-x0).^2/2/sigma_x
       + (Y_Grid-y0).^2/2/sigma_y ));
elseif (strcmp(Parameter.Topo_name,'none') && ~
   WithTopo) || strcmp(Parameter.Topo_name,'off')
    Z_Grid = 0.*X_Grid;
else
    load(Parameter.Topo_name);
    Z_Grid = interp2(XTopo,YTopo,ZTopo,X_Grid,Y_Grid);
end

% saves setup
if Parameter.Save
    filestem = ['../+output/',Parameter.Foldername];
    if ~exist(filestem,'dir'); mkdir(filestem); end

    save(['../+output/',Parameter.Foldername,'/Setup.
       mat'])
end

% compute topography gradient
[Gradient_x,Gradient_y] = gradient(Z_Grid,resolution,
```

```
    resolution);

%convert time
maxtime         = Parameter.maxtime*60; %[min] => [s]
decision_step   = round(Parameter.decision_time/
    Parameter.dt);

%--------------------------------------
% create starting area map for agents
%--------------------------------------
StartingList(find(StartingList(:,1)>=xmax),:) = []; %
    if set fully outside domain: remove it!
StartingList(find(StartingList(:,3)>=ymax),:) = []; %
    if set fully outside domain: remove it!
StartingList(find(StartingList(:,2)>xmax),2) = xmax; %
    adjust to domain boundary
StartingList(find(StartingList(:,4)>ymax),2) = ymax; %
    adjust to domain boundary

StartArea = logical(X_Grid*0);
% add buildings to map
for i=1:size(StartingList,1)
     StartArea(X_Grid>=StartingList(i,1) & X_Grid<=
         StartingList(i,2) & Y_Grid>=StartingList(i,3) &
          Y_Grid<=StartingList(i,4)) = true;
end

%--------------------------------------
% create boundary map for later use
%--------------------------------------
BoundaryMap = zeros(size(yvec,2),size(xvec,2));
BoundaryMap(1,:)=1; BoundaryMap(size(yvec,2),:)=1;
    BoundaryMap(:,1)=1; BoundaryMap(:,size(xvec,2))=1;

%--------------------------------------
% create building map for later use
%--------------------------------------
BuildingList(find(BuildingList(:,1)>=xmax),:) = []; %
    if building fully outside domain: remove it!
BuildingList(find(BuildingList(:,3)>=ymax),:) = []; %
    if building fully outside domain: remove it!
BuildingList(find(BuildingList(:,2)>xmax),2) = xmax; %
    adjust building to domain boundary
BuildingList(find(BuildingList(:,4)>ymax),2) = ymax; %
    adjust building to domain boundary
```

```
BuildingMap = logical(X_Grid*0);  BuildingMap_sp =
   BuildingMap;
% add buildings to map
for i=1:size(BuildingList,1)
    BuildingMap(X_Grid>=BuildingList(i,1) & X_Grid<=
       BuildingList(i,2) & Y_Grid>=BuildingList(i,3) &
        Y_Grid<=BuildingList(i,4)) = true;
    %for shortest path formulation:
    BuildingMap_sp(X_Grid>=(BuildingList(i,1)-
       Parameter.Enlarge) & X_Grid<=(BuildingList(i,2)
       +Parameter.Enlarge) ...
        & Y_Grid>=(BuildingList(i,3)-Parameter.Enlarge
            ) & Y_Grid<=(BuildingList(i,4)+Parameter.
            Enlarge) ) = true;
end

%-----------------------------------------------------------------

% create exit map for later use and compute center
   point of exits
%-----------------------------------------------------------------

ExitList(find(ExitList(:,1)>=xmax),:)   = []; %if exit
    fully outside domain: remove it!
ExitList(find(ExitList(:,3)>=ymax),:)   = []; %if exit
    fully outside domain: remove it!
ExitList(find(ExitList(:,2)>xmax),2)    = xmax; %
   adjust exit to domain boundary
ExitList(find(ExitList(:,4)>ymax),2)    = ymax; %
   adjust exit to domain boundary

ExitMap = logical(X_Grid*0);
for i=1:size(ExitList,1)
    ExitMap(X_Grid>=ExitList(i,1) & X_Grid<=ExitList(i
       ,2) & Y_Grid>=ExitList(i,3) & Y_Grid<=ExitList(
       i,4)) = true;
end

%-----------------------------------------------------
% create flood map
%-----------------------------------------------------
%compute height
Z_flood_deep = Z_flood - dangerousDepth;

%create floodmap
FloodMap = logical(X_Grid*0); FloodMap_deep = FloodMap
```

```
   ;
FloodHeightMap = ones(size(Z_Grid))*Z_flood;
FloodMap(FloodHeightMap>Z_Grid) = 1;
FloodHeightMap_deep = ones(size(Z_Grid))*Z_flood_deep;
FloodMap_deep(FloodHeightMap_deep>Z_Grid) = 1;

% initialize agents
nagent  = Parameter.nagent;
AGENT   = InitializeAgents(nagent,Parameter);

% create random agent distribution
if strcmp(Parameter.AgentSetup,'random')
    AGENT   = CreateInitialAgentDistribution(nagent,
       AGENT,X_Grid,Y_Grid,BuildingMap,BoundaryMap,
       StartArea,ExitMap);
elseif strcmp(Parameter.AgentSetup,'given')
    cell_array = num2cell(AgentX);
    [AGENT(1:nagent).LocX]      = cell_array{:};
    cell_array = num2cell(AgentY);
    [AGENT(1:nagent).LocY]      = cell_array{:};
elseif strcmp(Parameter.AgentSetup,'load')
    load(Parameter.AgentLocationFile);
    cell_array = num2cell(AgentX);
    [AGENT(1:nagent).LocX]      = cell_array{:};
    cell_array = num2cell(AgentY);
    [AGENT(1:nagent).LocY]      = cell_array{:};
end

%-----------------------------------------------------
% building locations
%-----------------------------------------------------
x_Buildings = X_Grid(BuildingMap);
y_Buildings = Y_Grid(BuildingMap);

%-----------------------------------------------------
% compute forces from buildings (static)
%-----------------------------------------------------
[ArchForce,ArchD,ArchDirX,ArchDirY] =
   ArchitectureForceV2(X_Grid,Y_Grid,BuildingMap,
   Parameter,resolution);




%-----------------------------------------------------
% compute shortest path to exit
```

```
%--------------------------------------------------
BuildingMap_boundary = zeros(size(BuildingMap));
   BuildingMap_boundary(BuildingMap~=BuildingMap_sp) =
    1;
if (~DirectExitPath && ~WithTopo)
    % compute shortest path without topography with
       fast marchng algorithm
    [Dgradx,Dgrady,D_orig] = ComputeShortestPathGlobal
       (FloodMap,FloodMap_deep,BuildingMap,
       BuildingMap_boundary,ExitMap,X_Grid,Y_Grid,
       Parameter);
elseif (~DirectExitPath && WithTopo)
    % compute shortest path without topography with
       fast marchng algorithm
    [~,~,D_orig] = ComputeShortestPathGlobal(FloodMap,
       FloodMap_deep,BuildingMap,BuildingMap_boundary,
       ExitMap,X_Grid,Y_Grid,Parameter);
    % compute shortest path with topography with fast
       marchng algorithm
    [Dgradx,Dgrady,D_orig] =
       ComputeShortestPathGlobalTopo(FloodMap,
       FloodMap_deep,BuildingMap,BuildingMap_boundary,
       ExitMap,X_Grid,Y_Grid,Z_Grid,D_orig,Gradient_x,
       Gradient_y,Parameter);
elseif DirectExitPath
    % compute exit direction directly
    [Dgradx,Dgrady] = ComputeShortestPathGlobalDirect(
       BuildingMap,ExitMap,X_Grid,Y_Grid,Parameter.v0,
       Parameter.resolution);
end

%--------------------------------------------------
% plot setup
%--------------------------------------------------
if PlotSetup
    % plot setup
    figure(1),clf
    set(cla,'FontSize',Plotting.FontSize)
    hold on
    % plot buildings
    PlotBuildings(BuildingList,'r','');
    PlotBuildings(ExitList,'g','Exit');
    % plot agents
    PlotAgents(AGENT,Plotting);
    axis equal
    axis([min(X_Grid(:)) max(X_Grid(:)) min(Y_Grid(:))
```

```
        max(Y_Grid(:))])
    box on
    title('time = 0.00 min')
    xlabel('x [m]')
    ylabel('y [m]')
end

%--------------------------------------------------
% save data (INITIAL SETUP)
%--------------------------------------------------
if Parameter.Save
    filestem = ['../+output/',Parameter.Foldername];
    if ~exist(filestem,'dir'); mkdir(filestem); end
    filename_full = [filestem,'/',Parameter.Foldername
       ,'_',num2str(0,'%5.6d')];
    save(filename_full,'AGENT')
end

%setup analysis variable
%Analysis = [num/name startPosX startPosY ExitTime
   Status]
%                                        Status: :   1:
     'alive' still running
%                                                    2:
     'survived' reached exit
%                                                    3:
     'killed' e.g. by flood
Analysis        = zeros(nagent,5)*NaN;
Analysis(:,1)   = [AGENT.name]';
Analysis(:,2)   = [AGENT.LocX]';
Analysis(:,3)   = [AGENT.LocY]';
Analysis(:,5)   = 1;



%=========================================================================

% time loop
time=0; itime=0;
while (time <= maxtime && size(AGENT,2)>0)
    time = time+Parameter.dt;     %actual time [s]
    itime = itime+1;    %nr. timesteps
    disp('****************************************')
    disp(['timestep ',num2str(itime),':    time = ',
       num2str(time/60),' min'])
```

```
if sum(isnan([AGENT.LocX]))
    error('NaN');
end

%-----------------------------------------------------

% interpolate z-level Agent
%-----------------------------------------------------

agent_Locz = interp2(X_Grid,Y_Grid,Z_Grid,[AGENT.
   LocX],[AGENT.LocY],'*linear');
dummy                       = num2cell(agent_Locz)
   ;
[AGENT(1:nagent).LocZ]      = dummy{:};

%-----------------------------------------------------

% compute flooding
%-----------------------------------------------------

if WithFlood
    %compute height
    Z_flood  = Z_flood + dzdt_flood*Parameter.dt;
    Z_flood_deep = Z_flood - dangerousDepth;

    %create floodmap
    FloodMap = logical(X_Grid*0); FloodMap_deep =
       FloodMap;
    FloodHeightMap = ones(size(Z_Grid))*Z_flood;
    FloodMap(FloodHeightMap>Z_Grid) = 1;
    FloodHeightMap_deep = ones(size(Z_Grid))*
       Z_flood_deep;
    FloodMap_deep(FloodHeightMap_deep>Z_Grid) = 1;

    % compute forces from flood
    if (Z_flood>min(min(Z_Grid))) % for shallow
       part
        [FloodForce,~,FloodDirX,FloodDirY] =
           f_FloodForce(X_Grid,Y_Grid,FloodMap,
           Parameter,resolution);
        % compute forces from flood (2nd part) on
           all agents
        % and interpolate it to the agent
        [FxSocialFlood,FySocialFlood] =
           ComputeSocialForcesStatic_flood(AGENT,
           X_Grid,Y_Grid,FloodForce,FloodDirX,
```

```
            FloodDirY,Parameter);
        dummy                                   =
            num2cell(FxSocialFlood);
        [AGENT(1:nagent).FxSocialFlood]         =
            dummy{:};
        dummy                                   =
            num2cell(FySocialFlood);
        [AGENT(1:nagent).FySocialFlood]         =
            dummy{:};
    end
    if (Z_flood_deep>min(min(Z_Grid))) % for deep
       part
        [FloodForce_deep,~,FloodDirX_deep,
            FloodDirY_deep] = f_FloodForce(X_Grid,
            Y_Grid,FloodMap_deep,Parameter,
            resolution);
        % compute forces from flood (2nd part) on
            all agents
        % and interpolate it to the agent
        [FxSocialFlood_deep,FySocialFlood_deep] =
            ComputeSocialForcesStatic_flood(AGENT,
            X_Grid,Y_Grid,FloodForce_deep,
            FloodDirX_deep,FloodDirY_deep,Parameter
            );
        dummy                                   =
            num2cell([AGENT(1:nagent).FxSocialFlood
            ] + FxSocialFlood_deep);
        [AGENT(1:nagent).FxSocialFlood]     =
            dummy{:}; %add to shallow flood force
        dummy                                   =
            num2cell([AGENT(1:nagent).FySocialFlood
            ] + FySocialFlood_deep);
        [AGENT(1:nagent).FySocialFlood]     =
            dummy{:}; %add to shallow flood force
    end
end

%--------------------------------------------------

% compute kdtree of agents for later use
%--------------------------------------------------

ReferencePoints          = zeros(nagent,2);
ReferencePoints(:,1)     = [AGENT(:).LocX];
ReferencePoints(:,2)     = [AGENT(:).LocY];
```

```
% generate tree
tree =   kdtree(ReferencePoints);

%--------------------------------------------------

% compute forces from buildings (2nd part) on all
   agents
% and interpolate it to the agent
%--------------------------------------------------

if Parameter.SocialForces
    [FxSocialWalls,FySocialWalls] =
       ComputeSocialForcesStatic(AGENT,X_Grid,
       Y_Grid,ArchForce,ArchDirX,ArchDirY,
       Parameter);
    dummy                                  =
       num2cell(FxSocialWalls);
    [AGENT(1:nagent).FxSocialWalls]        = dummy
       {:};
    dummy                                  =
       num2cell(FySocialWalls);
    [AGENT(1:nagent).FySocialWalls]        = dummy
       {:};
else
    [AGENT(1:nagent).FxSocialWalls]        = deal
       (0);
    [AGENT(1:nagent).FySocialWalls]        = deal
       (0);
end

%--------------------------------------------------

% compute direction field to exits on all agents
% (just interpolate the precomputed field to the
   agents)
%--------------------------------------------------

if (~DirectExitPath && WithAgents)
    if (mod(itime,decision_step)==0 || itime==1)
        [Dgradx,Dgrady] =
           ComputeShortestPathGlobalWithAgents(
           BuildingMap,BuildingMap_boundary,...
            ExitMap,X_Grid,Y_Grid,D_orig,Dgradx,
               Dgrady,Gradient_x,Gradient_y,AGENT,
               nagent,Parameter);
    end
```

```
    elseif (~DirectExitPath && WithAgents && WithFlood
       )
        if (mod(itime,decision_step)==0 || itime==1)
            [Dgradx,Dgrady] =
               ComputeShortestPathGlobalWithAgentsFlood
               (FloodMap,FloodMap_deep,BuildingMap,
               BuildingMap_boundary,...
                ExitMap,X_Grid,Y_Grid,D_orig,Dgradx,
                   Dgrady,Gradient_x,Gradient_y,AGENT,
                   nagent,Parameter);
        end
    end

    xExitDirAgents = interp2(X_Grid,Y_Grid,Dgradx,[
       AGENT.LocX],[AGENT.LocY],'*linear');
    yExitDirAgents = interp2(X_Grid,Y_Grid,Dgrady,[
       AGENT.LocX],[AGENT.LocY],'*linear');

%     % normalize direction vector
%     dirtot         = sqrt(xExitDirAgents.^2+
   yExitDirAgents.^2);
%     xExitDirAgents = xExitDirAgents./dirtot;
%     yExitDirAgents = yExitDirAgents./dirtot;

    dummy = num2cell(xExitDirAgents);
    [AGENT(1:nagent).xExitDir]       = dummy{:};
    dummy = num2cell(yExitDirAgents);
    [AGENT(1:nagent).yExitDir]       = dummy{:};


    %--------------------------------------------------

    % agent loop
    iagent=0; nagent2 = nagent;
    for iagent = 1:nagent2
        x_agent         = AGENT(iagent).LocX;
        y_agent         = AGENT(iagent).LocY;
        agent_size      = AGENT(iagent).Size;
        velx_agent      = AGENT(iagent).VelX;
        vely_agent      = AGENT(iagent).VelY;

        %-------------------------------------------------

        % check if the agent is outside the domain
        %-------------------------------------------------
```

```
if x_agent>xmax || x_agent<xmin || y_agent>
   ymax || y_agent<ymin
   error('fc: stupid agent outside domain!')
end

%-------------------------------------------------

% get the agents that are in the "individual
   box" and compute the
% distance to them
%-------------------------------------------------

[AGENT,x_others,y_others,others_size] =
   GetSurroundingAgents(iagent,AGENT,tree);

[Normal,Tangent,DistanceToAgents,num_others] =
    ComputeDistanceToAgents(x_agent,y_agent,
   agent_size,x_others,y_others,others_size);

if num_others >0
    % find agents that are too close
    indTooClose     = find(DistanceToAgents
       >=0);

    %-----------------------------------------------------

    % compute social forces from other agents
       and apply a weighting
    % function to simulate that agents only
       have a reduced field of
    % vision
    %-----------------------------------------------------

    if Parameter.SocialForces
        [FxAgentsSocial,FyAgentsSocial] =
           ComputeSocialForcesDynamic(
           Parameter,DistanceToAgents,Normal);
    else
        FxAgentsSocial = 0;
        FyAgentsSocial = 0;
    end

    %-----------------------------------------------------

    % compute physical forces from other
       agents
```

```
    %--------------------------------------------------

    if (~isempty(indTooClose) && Parameter.
       PhysicalForces)
        surr_agents     = AGENT(iagent).
           SurroundingAgents;
        AgentTooClose   = surr_agents(
           indTooClose);
        DistTooClose    = DistanceToAgents(
           indTooClose);

        velx_others = AGENT(AgentTooClose).
           VelX;
        vely_others = AGENT(AgentTooClose).
           VelY;

        [FxPhysAgents,FyPhysAgents] =
           ComputePhysicalForceAgents(
           velx_agent,vely_agent,velx_others,
           vely_others,Parameter,DistTooClose,
           Normal(indTooClose,:),Tangent(
           indTooClose,:));
    else
        FxPhysAgents = 0;
        FyPhysAgents = 0;
    end
else
    FxAgentsSocial = 0;
    FyAgentsSocial = 0;
    FxPhysAgents = 0;
    FyPhysAgents = 0;
end

%--------------------------------------------------

% compute physical forces from walls
%--------------------------------------------------

if Parameter.PhysicalForces
    [FxPhysWall,FyPhysWall] =
       ComputePhysicalForceWalls(x_agent,
       y_agent,agent_size,velx_agent,
       vely_agent,x_Buildings,y_Buildings,
       Parameter);
else
    FxPhysWall = 0;
```

```
        FyPhysWall = 0;
    end

    %-----------------------------------------------------

    % assign forces to structure
    %-----------------------------------------------------

    AGENT(iagent).FxPhysAgents   = sum(
       FxPhysAgents);
    AGENT(iagent).FyPhysAgents   = sum(
       FyPhysAgents);
    AGENT(iagent).FxPhysWall     = sum(FxPhysWall)
       ;
    AGENT(iagent).FyPhysWall     = sum(FyPhysWall)
       ;

    % add some random noise on the social force
       from other agents
    AGENT(iagent).FxSocialAgents = sum(
       FxAgentsSocial)*(1+ Parameter.pert_social
       *(-0.5+rand(1)) );
    AGENT(iagent).FySocialAgents = sum(
       FyAgentsSocial)*(1+ Parameter.pert_social
       *(-0.5+rand(1)) );
end

%-------------------------------------------------------

% compute exit force
%-------------------------------------------------------

[AGENT] = ComputeExitForce(AGENT,Parameter,nagent)
   ;

%-------------------------------------------------------

% check if agents in flood
%-------------------------------------------------------

wet = [AGENT( [AGENT.LocZ]<=Z_flood ).num];
if ~isempty(wet)
    for iwet=1:size(wet,2)
        AGENT(wet(1,iwet)).Status = 2; %wet
        AGENT(wet(1,iwet)).VMax = Parameter.
           FloodSpeed; %decrease max. speed by
```

```
            half VMax
    end
end

%-------------------------------------------------------

% move agents
%-------------------------------------------------------

[AGENT] = MoveAgents(AGENT,X_Grid,Y_Grid,
   Gradient_x,Gradient_y,Parameter.dt,nagent,
   Parameter);

%-------------------------------------------------------

% check if agents are inside walls and move them
   out
%-------------------------------------------------------

AGENT = CheckAgentsInBuildings(AGENT,BuildingList,
   X_Grid,Y_Grid,ArchDirX,ArchDirY,ArchD);

%-------------------------------------------------------

% remove successfull/dead agents
%-------------------------------------------------------

%those who arrived in the exits
for i=1:size(ExitList,1)
    successfull = [AGENT(  [AGENT.LocX]>=ExitList(
       i,1) & [AGENT.LocX]<=ExitList(i,2) ...
        & [AGENT.LocY]>=ExitList(i,3) & [AGENT.
           LocY]<=ExitList(i,4) ).num];
    %save time of agents exit
    Analysis([AGENT(successfull).name],4) = time;
       %in [s]
    Analysis([AGENT(successfull).name],5) = 2; %
       change status to 'survived'

    AGENT(successfull) = []; %remove agents
end

%remove agents outside model domain
AGENT(   [AGENT.LocX]>xmax | [AGENT.LocX]<xmin ...
    | [AGENT.LocY]>=ymax | [AGENT.LocY]<ymin  ) =
       [];
```

```
%remove agents in deep water
drowned = [AGENT( [AGENT.LocZ]<=Z_flood_deep ).num
   ];
if ~isempty(drowned)
    %save time of agents exit
    Analysis([AGENT(drowned).name],4) = time; %in
       [s]
    Analysis([AGENT(drowned).name],5) = 3; %change
        status to 'drowned'

    AGENT(drowned) = []; %remove agents
end


nagent = size(AGENT,2); %update number of agents
   after removing some of them
cell_array = num2cell(1:nagent); [AGENT(1:nagent).
   num] = cell_array{:}; %update correct numbering
    from 1:nagent


%-------------------------------------------------------

% save data
%-------------------------------------------------------

if Parameter.Save && mod(itime,Parameter.
   SaveTimeStep)==0
    filestem = ['../+output/',Parameter.Foldername
       ];
    if ~exist(filestem,'dir'); mkdir(filestem);
       end

    filename_full = [filestem,'/',Parameter.
       Foldername,'_',num2str(itime,'%5.6d')];

    save(filename_full,'AGENT')
end

%-------------------------------------------------------

% plot
%-------------------------------------------------------
```

```
    if (PlotEvolution && mod(itime,Parameter.
        PlotTimeStep)==0)

        figure(1),clf
        set(cla,'FontSize',Plotting.FontSize)
        hold on
%        pcolor(X_Grid,Y_Grid,Z_Grid),shading flat,
    colorbar
%        quiver(X_Grid,Y_Grid,Dgradx,Dgrady,'b')
        contour(X_Grid,Y_Grid,double(FloodMap),[1 1],'
            b-'); %colorbar; colormap('winter');
        contourf(X_Grid,Y_Grid,double(FloodMap_deep)
            ,[1 1],'r-'); %colorbar; colormap('bone');
        % plot buildings
        PlotBuildings(BuildingList,'k','');
        PlotBuildings(ExitList,'g','Exit');
        % plot agents
        PlotAgents(AGENT,Plotting);

        % quiver([AGENT(1:nagent).LocX],[AGENT(1:
            nagent).LocY],[AGENT(1:nagent).xExitDir],[
            AGENT(1:nagent).yExitDir],'r')

        % quiver([AGENT.LocX],[AGENT.LocY],[AGENT.DirX
            ],[AGENT.DirY],'r-')
        axis equal
        axis([min(X_Grid(:)) max(X_Grid(:)) min(Y_Grid
            (:)) max(Y_Grid(:))])
        box on
        title(['time = ',num2str(time,'%.2d'),' s'])
        xlabel('x [m]')
        ylabel('y [m]')
        %axis([AGENT(1).LocX-10 AGENT(1).LocX+10 AGENT
            (1).LocY-10 AGENT(1).LocY+10])

        pause(0.01)
    end
end

% saves analysis
if Parameter.Save
    save(['../+output/',Parameter.Foldername,'/
        Analysis.mat'],'Analysis')
end

%=========================================================================
```

[