Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB
Project Report

Document Version: 1.0
Group Name: Rieparjo
Group Members: Christoph Rieper and Benjamin Sunarjo

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Christoph Rieper

Benjamin Sunarjo

Contents

Abstract

Individual contributions

1 Introduction

Agent-based models can provide a easily implementable way to study complex systems. As Helbing et al. (1997) have shown, many aspects of pedestrian motion, such as the formation of trail systems in green areas, can be reproduced using a relatively simple "active walker" model that takes into account the attractiveness of terrain and feedback on the terrain as it is walked upon. In the current project, we plan to apply such an active walker model to real landscapes and compare the results to existing road systems.

We attempt to answer the question: is the active walker model able to predict reasonable pathways between neighboring villages in real landscapes? Here, "reasonable" will be evaluated first in a qualitative sense. Second, a energy function will be defined based on the distance traveled horizontally and vertically, where a minimal energy function is most reasonable.

In a second step, we will determine the influence of landscape slope on trail formation, under the assumption that modern roads are situated where historically trails used to go through. We will compare generated paths to current road networks at two test sites to answer the questions: How does trail formation change with increasing landscape slope? Do the formed paths fit to current road networks?

Theoretical work by Helbing et al. (1997) has previously been implemented in an agent-based model by Pfefferle & Pleschko (2010). We will base our investigation of the above research questions on this model, making adjustments where necessary. We will use topographical data from swisstopo.admin.ch with an emphasis on 1. determining reasonable model parameters and 2. comparing modeled trails to existing road systems. Two test sites are proposed, one in an mountainous region in St. Moritz, the other in the Swiss lowlands near Friburg (Figure 1). These two test sites provide very different types of terrain on which to study the problem of trail formation.

We expect to find that the smaller roads correspond more closely to results generated by the active walker model, while larger cantonal roads, being further removed from their trail origins, should correspond less with model results. We further expect increasingly mountainous terrain to tightly constrain possible routes: we expect closer correlation between road systems and generated model results in mountainous regions than in the lowlands, since there are less possibilities for taking a route with low associated energy cost.

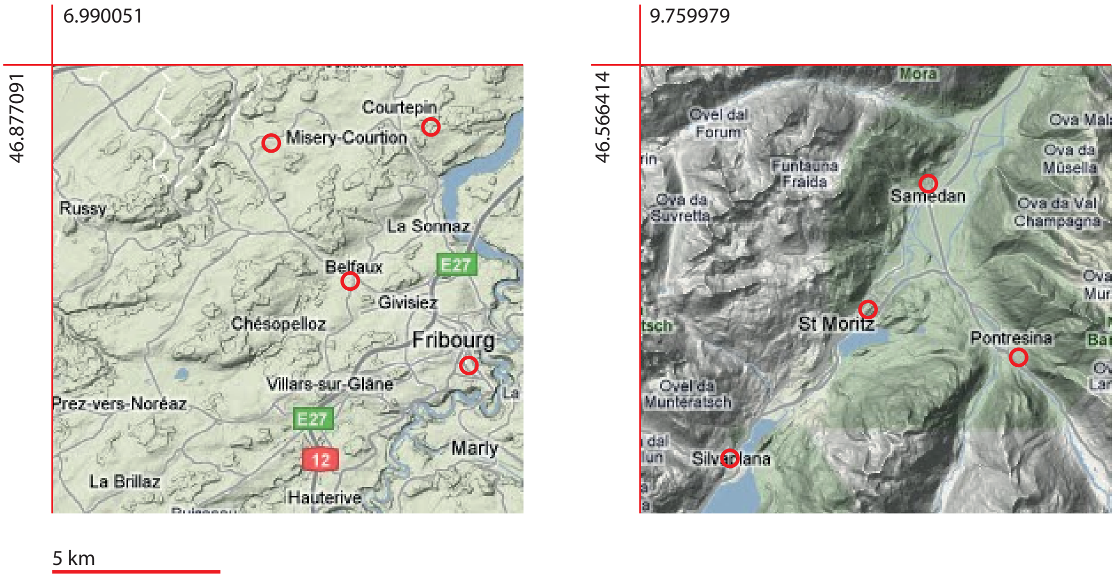

Figure 1. Two test sites, one mountainous one flat.

1.1 site

2 Description of the Model

Our studies are based on an active walker model developed by Helbing et al. (1997). Helbing et al. (1997) used this agent-based model to explain footpath formation in green areas in cities.

In comparison to a normal pedestrian model, an active walker model also takes the interactions of the pedestrians and the terrain they walked upon into account. This means that the pedestrians change the landscape they walk upon and the changed landscape influences again the pedestrians movement. A second important characteristic of the model, which influences the direction of walking of the pedestrians, is the attractiveness of a trail segment. It is a property of each point in the terrain, which describes how interesting it is to go to this certain place. In other words, how good the prospects are on this place for further walking. It is the element of our model which handles the effect of human orientation and is described later on in more detail.

As the model consists of three major components (the ground, the attractiveness of a trail segment and the pedestrians), all three are described separately. The ground and the influence of the pedestrians on it is described in Sec. 2.1. Sec. 2.2 explains how the attractiveness of walking is computed and Sec. 2.3 illustrates how the pedestrians walking direction is determined.

2.1 Ground Structure

Our landscape is represented by a function $G(r,t)$, called comfort of walking, with r for the position in the plane and t for time. As it is a property of our plane it is defined on each point. High values of G stand for trails, i.e. places where many people walked upon, while low values of G stand for places where fewer people passed by. Every time a pedestrian walks on o certain point of the plain it changes the comfort of walking there. This is because pedestrians trample down the vegetation. This is described by

$$I(r)[1 - \frac{G(r,t)}{G_{max}(r)}] \, , \tag{1}$$

where $I(r)$ stands for the intensity of the footprint and $G_{max}(r)$ for the maximal value of the comfort of walking at a certain place (i.e. the maximal value a place can be trampled "up"). The expression in the brackets of Eq. 1 account for the saturation effect, so that the impact of the footprints decreases when there are more people walking on a place until the maximal value is reached.

As the vegetation can be trampled down it can also regrow. This effect is expressed by

$$\frac{1}{T(r)}[G_0(r) - G(r,t)] \tag{2}$$

where $G_0(r)$ stands for the natural ground condition and $T(r)$ for the durability of the trails. The bigger the durability $T(r)$ the slower the ground goes back to natural conditions $G_0(r)$.

Finally the change of the comfort of walking by time due to the walking pedestrians and the regrowth of the vegetation can be expressed as

$$\frac{dG(r,t)}{dt} = I(r)[1 - \frac{G(r,t)}{G_{max}(r)}] \sum_{\alpha} \delta(r - r_{\alpha}(t)) + \frac{1}{T(r)}[G_0(r) - G(r,t)] \tag{3}$$

with α for the set of pedestrians and $\delta(r - r_{\alpha}(t))$ standing for the Dirac delta function, which is 1 if $r = r_(\alpha)$ and 0 in all other cases and therefore only contributs if a pedestrian is on the actual position.

2.2 Attractiveness of Trail Segment

As mentioned above the attractiveness of a trail segment is a measure of how interesting a place is in manner of later onward walking. It is defined for every place and depends on the comfort of walking of its surrounding, where the influence of the surround decreases with distance from the place. The attractiveness of a trail segment is called trail potential and is defined as

$$V_{tr}(r_t, t) = \int_P G(r, t) e^{\frac{-|r-r_t|}{\sigma(r_t)}} dP \tag{4}$$

where r_t stands for the position the trail potential is computed for, P for the plain and $\sigma(r_t)$ for the visibility. The visibility controls how fast the influence of the surrounding decreases. The higher the visibility the slower the influence of the surrounding decreases. Furthermore high values of the trail potential stand for a high attractiveness of a trail segment and vice versa.

2.3 Pedestrians Walking Direction

In the model every pedestrian has a starting point and a destination. When the pedestrians walking direction is determined two vectors decide about the final walking direction. One vector is the vector which points towards the pedestrians destination. It is given by the unit vector

$$e_\alpha^1(r_\alpha, t) = \frac{d_\alpha - r_\alpha}{|d_\alpha - r_\alpha|} \tag{5}$$

where d_α is the position of the pedestrians destination.The other vector which decides about the pedestrians walking direction is the vector which points into the direction of highest increase of the trail potential. It can be expressed in the normalized form as

$$e_\alpha^2(r_\alpha, t) = \frac{\nabla V_{tr}(r_\alpha, t)}{|\nabla V_{tr}(r_\alpha, t)|}. \tag{6}$$

Combining Eq. 5 and 6 and introducing a new variable ρ that controls the relative importance of the two vectors leads to the final walking direction

$$e_\alpha(r_\alpha, t) = \rho \cdot e_\alpha^1(r_\alpha, t) + e_\alpha^2(r_\alpha, t) = \rho \cdot \frac{d_\alpha - r_\alpha}{|d_\alpha - r_\alpha|} + \frac{\nabla V_{tr}(r_\alpha, t)}{|\nabla V_{tr}(r_\alpha, t)|}. \tag{7}$$

For value of $\rho > 1$ the destination vector gets more important and for value $\rho < 1$ the direction of the highest increase of the trail potential prevails.

3 Description of the Path-Evaluation Function

To evaluate the paths taken by the pedestrians a function was defined to judge if a path is reasonable or not. As described by Kölbl & Helbing (2003) humans try to minimize their cost of travel. Cost of travel in our case is travel-time. Therefore we developed a function which calculates the time it needs to walk a certain path.

Assuming a horizontal speed $u_{horiz}(G(r_\alpha, t))$ which scales with comfort of walking and a constant vertical speed u_{vert} the travel-time is given by

$$T = \frac{s_{horiz}}{u_{horiz}(G(r_\alpha, t))} + \frac{s_{vert}}{u_{vert}}, \tag{8}$$

where s_{vert} is the uphill travelled distance and s_{horiz} the horizontally traveled distance. In more detail the horizontal speed u_{horiz} scales between a minimal horizontal speed $u_{horiz,min}$ and a maximal horizontal speed $u_{horiz,max}$ depending on how strongly the path is trampled down. This is expressed as

$$u_{horiz}(r_\alpha, t) = u_{horiz,min} + (u_{horiz,max} - u_{horiz,min}) \cdot \frac{G(r_\alpha, t) - G_0(r_\alpha, t)}{G_{max}(r_\alpha, t) - G_0(r_\alpha, t)}. \tag{9}$$

In Tab. 1 values of $u_{horiz,min}$, $u_{horiz,max}$ and u_{vert} which are used in the path-evaluation function can be found.

Table 1. Vertical and horizontal speed used in path-evaluation function.

$u_{horiz,min}$	$u_{horiz,max}$	u_{vert}
4000 $m\,h^{-1}$	6000 $m\,h^{-1}$	500 $m\,h^{-1}$

4 Implementation

The active walker model was previously implemented by Pfefferle & Pleschko (2010). We further developed the model by adding the path-evaluation function and the interface to work with real elevation data to its functionalities. The implementation by Pfefferle & Pleschko (2010) is described in their report, while our contribution is described here.

4.1 Interface for real Elevation Data

The elevation data provided by swisstopo.admin.ch (5×5 km for each site) came in a resolution of 25×25 m. To use this elevation data in our model first the resolution had to to be lowered to 500×500 m for computational reasons and then the data had to bee adjusted to work in our model.

4.2 Path Class

As our

Figure 2. The road structure at the two study sites. Data taken from Swisstopo illustrates the differences. The weight of the lines is derived from road class order (highways and 1. order roads are heavier in weight than higher order roads) and does not necessarily correlate to throughput.

5 Simulation Results and Discussion

In our analysis of the path formation process using our active walker model, the biggest uncertainty concerned the values of the various independent parameters (durability and intensity of trails, visibility, relative importance of the destination vector). No literature values could be found for these values (they are dependent on model specifics such as the model scale). Since our initial hypotheses cannot be conclusively addressed without first dealing with the issue of model parameters, we first discuss the effects parameter variation has on model results, then look in detail at the generated path structures. Finally, we compare our results with existing road structures.

5.1 Parameter variation

visibility

importance

durability

intensity

5.2 Path structure

5.3 Existing road structure

qualitative differences between the two sites. Road structure at the St. Moritz site shows close similarity to the structure of alpine rivers,

6 Summary and Outlook

We found: -it's most important to determine correct parameter values -there are qualitative differences between the two study sites.

7 References

Helbing, D., Keltsch, J., & Molnár, P. (1997). Modelling the evolution of human trail systems. *Nature*, 388(3).

Kölbl, R. & Helbing, D. (2003). Energy laws in human travel behaviour. *New Journal of Physics*, 5(48).

Pfefferle, J. & Pleschko, N. (2010). Simulation of human trail systems. Project Report for Lecture "Modelling and Simulating Social Systems with MATLAB".

8 Appendix: Matlab Code

batch.m

```

% location of cities on original grid (25x25m)
stm_p = [454 620; 567 452; 674 638; 528 542; 328 785; 623 372];
fri_p = [667 503; 596 534; 693 607; 599 227; 612 459; 546 417; 255 223; ...
    662 399; 412 655; 510 689; 523 514; 418 366; 418 570; 267 669];

% location of cities on reduced grid (500x500m)
stm_p = [23 31;29 23;34 32;27 28;17 40;32 19];
fri_p = [34 26;30 27;35 31;30 12;31 23;28 21;13 12;34 20;21 33;26 35;27 ...
    26;21 19;21 29;14 34];


%dur = 25;                          % Durability
%inten = 10;                        % Intensity
%vis = 0.3, 1, 4;                           % Visability
%importance = 1.6;
%location = 'fri'
%numframes = 100

for vis=[0.3 1 4]
    for dur = [5 25 50]
        for inten = [5 10 30]
            for importance = [0.5 1 1.6]
                smDriver(dur, inten, vis, importance, 'fri', 100);
                smDriver(dur, inten, vis, importance, 'stm', 100);
            end
        end
    end
end
```

Path.m

```
classdef Path < handle
    %PATH our path class

    properties(SetAccess = public)
        coordinates;
        type;
        time;
        timeOfArrival;
        relativeGround;
    end
```

```
    methods

        function obj = Path(ped,entryP,speed,aPlain,aTime)
            % takes deleted pedestrian and generates its path with
            % properties
            obj.coordinates = [ped.way; ped.destination];
            obj.relativeGround = [ped.relativeGround;...
                aPlain.relativePath(ped.destination(1),ped.destination(2))];
            obj.timeOfArrival = aTime;
            PathType(obj,entryP);
            PathTime(obj,speed,aPlain);
        end

        function PathType(obj,entryP)
            % checks which from the possible paths the pedestrian went

            % generate possible paths
            PossPathIndex = nchoosek(1:length(entryP),2);

            PossPath.origin = zeros(size(PossPathIndex));
            PossPath.destination = zeros(size(PossPathIndex));

            for i=1:size(PossPathIndex,1)
                PossPath.origin(i,:) = entryP(PossPathIndex(i,1),:);
                PossPath.destination(i,:) = entryP(PossPathIndex(i,2),:);
            end

            % check which from the possible paths the path is
            for i=1:size(PossPathIndex,1)

                if ((obj.coordinates(1,:)==PossPath.origin(i,:)|...
                        obj.coordinates(1,:)==PossPath.destination(i,:))&...
                        (obj.coordinates(end,:)==PossPath.origin(i,:)|...
                        obj.coordinates(end,:)==PossPath.destination(i,:)))
                  obj.type=i;
                end

            end
        end

        function PathTime(obj,speed,aPlain)
           % calculates the time it takes the pedestrian to walk the path

           path.horiz = obj.coordinates;

           % extracting height info from elevation model
           path.vert = zeros(size(path.horiz,1),1);
           for i=1:size(path.horiz,1)
               path.vert(i) = ...
                   aPlain.realGround(path.horiz(i,1),path.horiz(i,2));
```

```
            end



            % calculating horizontal and vertical distance from path
            % (vertical distance only taken if path goes uphill)

            dist_horiz = zeros(size(path.horiz,1)-1,1);
            dist_vert = zeros(size(path.horiz,1)-1,1);
            relativeGroundMean = zeros(size(path.horiz,1)-1,1);

            for i=1:size(path.horiz,1)-1

                % calculate horizontal distance
                delta_horiz = norm(path.horiz(i+1,:)-path.horiz(i,:));
                dist_horiz(i,1) = delta_horiz;

                % calculate horizontal distance
                delta_vert = path.vert(i+1)-path.vert(i);
                if delta_vert>=0
                    dist_vert(i,1) = delta_vert;
                else
                    dist_vert(i,1) = 0;
                end

                % calculate relative ground between the gridpoints
                relativeGroundMean(i,1) = ...
                    (obj.relativeGround(i+1,1)-obj.relativeGround(i,1))/2;

            end

            % scale horizontal distance with grid size and calculate walking
            % time

            speed.horizontal.real = speed.horizontal.min + ...
                (speed.horizontal.max-speed.horizontal.min)*relativeGroundMean;

            obj.time = ...
                sum(aPlain.gridSize*dist_horiz./speed.horizontal.real + ...
                dist_vert/speed.vertical);

        end

    end

end
```

Pedestrian.m

```
classdef Pedestrian < handle
    %PEDESTRIAN our pedestrian class

     properties(SetAccess = private )
        destination;
        way;
        relativeGround;
     end

    properties(SetAccess = public)
        position;
    end

    methods
        function obj = Pedestrian(entryP,aPlain)
            % generate new pedestrian and randomly choose origin and
            % destination from given entry points

            r = randperm(length(entryP));
            orig = entryP(r(1),:);
            dest = entryP(r(2),:);
            obj.way = orig;
            obj.destination = dest;
            obj.position = orig;
            obj.relativeGround = aPlain.relativePath(obj.position(1),...
                obj.position(2));
        end

        function set.position(obj,pos)
            % put pedestrian to position
            obj.position = pos;
        end

        function saveWay(obj,aPlain)
            % save way of pedestrian and relative strength of ground
            obj.way = [obj.way; obj.position];
            obj.relativeGround = [obj.relativeGround;...
                aPlain.relativePath(obj.position(1),obj.position(2))];
        end

        function val = isAtDestination(obj)
            % check if pedestrian arrived at destination
            val = (norm(obj.position - obj.destination)<2);
        end
    end
end
```

Plain.m

```
classdef Plain < handle
    %PLAIN Saves state of the plain

    properties(SetAccess = public)
        ground;          % The current ground structure
        initialGround;   % inital ground model
        groundMax;       % The maximum values of the walking comfort
        intensity;       % The footprint intensity
        durability;      % The durability of trails
        visibility;      % The visibility at each point
        realGround;      % real, not inverted, ground
        gridSize;        % grid size of plain in m
        relativePath;    % relative strength of path
    end

    methods
        function obj=Plain(aInitialGround,aGroundMax,aIntensity,...
                aDurability,aVisibility,aRealGround,aGridSize)
            initSize = size(aInitialGround);
            % Check if initialGround has same size as intensity and
            % durability matrix

            if((nnz(initSize == size(aIntensity)) == 2) &&...
                    (nnz(initSize == size(aDurability))==2) &&...
                    (nnz(initSize == size(aGroundMax))==2) &&...
                    (nnz(initSize == size(aVisibility))==2) &&...
                    (nnz(initSize == size(aRealGround))==2))

                obj.ground = aInitialGround;
                obj.groundMax = aGroundMax;
                obj.initialGround = aInitialGround;
                obj.intensity = aIntensity;
                obj.durability = aDurability;
                obj.visibility = aVisibility;
                obj.realGround = aRealGround;
                obj.gridSize = aGridSize;
            else
                error('PLAIN(): initialGround must be same size as ...
                    intensity and durability');
            end
        end

        function changeEnvironment(obj,pedestrians)
            % Changes the environment according to the positions of the
            % pedestrians
            [n m] = size(obj.ground);
            pedAt = sparse(n,m);

            for i=1:length(pedestrians)
                ped = pedestrians(i);
```

```
                pedAt(ped.position(1),ped.position(2)) = ...
                    pedAt(ped.position(1),ped.position(2)) + 1;
            end

            % Change the environment on each square of the plain
            for i=1:n
                for j=1:m
                    % Change the ground according to the formula
                    obj.ground(i,j) = obj.ground(i,j) + ...
                        1/obj.durability(i,j) * (obj.initialGround(i,j)-...
                        obj.ground(i,j)) + obj.intensity(i,j) * ...
                        (1-(obj.ground(i,j)/obj.groundMax(i,j))) * ...
                        pedAt(i,j);

                    % Check for the boundaries of the ground values
                    if(obj.ground(i,j) > obj.groundMax(i,j))
                        obj.ground(i,j) = obj.groundMax(i,j);
                    elseif(obj.ground(i,j) < obj.initialGround(i,j))
                        obj.ground(i,j) = obj.initialGround(i,j);
                    end
                end
            end

        end

        function val = isPointInPlain(obj,y,x)
            % Returns wheter or not a point (x,y) is in this plain
            val = (y>0 && x>0);
            val = val && y<=size(obj.ground,1) && x<=size(obj.ground,2);

        end

        function MakeRelativePath(obj)
            % Calculates relative path strength (1 = maximal path)
            obj.relativePath = (obj.ground-obj.initialGround)./...
                (obj.groundMax-obj.initialGround);
        end
    end

end
```

smDriver.m

```
function smDriver(dur, inten, vis, importance, site, numframes)
%SMDRIVER Sets up a simulation

f1 = figure('OuterPosition',[0 0 700 600]);
winsize = get(f1,'Position');
```

```
%numframes = 200;


% selecting elevation model
load elevation
if(strcmp(site,'stm'))
    elevation = stmoritz(1:1140,:); % for St. Moritz
    entryPoints = [23 31;29 23;34 32;27 28;17 40;32 19];
elseif(strcmp(site,'fri'))
    elevation = friburg(1:1100,1:1260); % for Friburg
    entryPoints = [34 26;30 27;35 31;30 12;31 23;28 21;13 12;34 20;21 ...
        33;26 35;27 26;21 19;21 29;14 34];
end


% resizing elevation model (original elevation dim must be multiple of 20)
elevation_re = zeros(size(elevation)/20);
[m n] = size(elevation);

for i=20:20:m
   for j=20:20:n
    elevation_re(i/20,j/20) = mean2(elevation(i-19:i,j-19:j));
   end
end

[m n] = size(elevation_re);


% Set the parameters
%dur = 25;                         % Durability
%inten = 10;                       % Intensity
%vis = 4;                          % Visability
%importance = 1.6;                 % Weight of the destination vector
speed.horizontal.min = 4000;     % min horizontal speed in m/h
speed.horizontal.max = 6000;     % max horizontal speed in m/h
speed.vertical = 500;            % vertical speed in m/h
gridSize = 500;                  % grid size of plain in m
pathMax = 100;                   % maximal value of a path


initialGround = max(max(elevation_re))-elevation_re; % inverting elevation
groundMax = initialGround + ones(m,n)*pathMax;
intensity = ones(m,n) * inten;
durability = ones(m,n) * dur;
visibility = ones(m,n) * vis;
elevation = elevation_re;

% create new plain with the specified values
myplain = Plain(initialGround,groundMax,intensity,durability,visibility,...
    elevation,gridSize);

```

```
% show the plain for input of the entry points
%pcolor(myplain.realGround);
%colormap(gray);
%shading interp;
%axis ij;
%entryPoints = ginput;
%entryPoints = floor([entryPoints(:,2) entryPoints(:,1)]);

% create a state machine with the specified plain
mysm = StateMachine(myplain);
mysm.importance = importance;
mysm.entryPoints = entryPoints;
mysm.speed = speed;
% make cell, where possible paths on Plain are later saved
noPossPaths = length(nchoosek(1:length(mysm.entryPoints),2));
mysm.pathsSorted = cell(1,noPossPaths);

% Do 'numframes' timesteps
C(1) = getframe(gcf);
for i=1:numframes

    % print every 20th timestep into a .png file
    if(mod(i,20)==0 && i >0)
        str = sprintf('images/im_%d_d%d_i%d_v%d_%d.png',...
            importance,dur,inten,vis,i);
        saveas(f1,str);
    end

    % compute a new transition in the state machine
    vtr = mysm.transition;

    % tell the StateMachine in which state it is in
    mysm.time = i;

    % display how many pedestrians are on the plane
    pedestrians = mysm.pedestrians;
    fprintf('Number of pedestrians: %d\n',length(pedestrians));

    % making the 3 subplots
    clf(f1);
    suptitle({[];[];['Grid:' num2str(m) 'x' num2str(n)];['Durability:'...
        num2str(dur) ' Visibility:' num2str(vis) ' '];[ 'Intensity:' ...
        num2str(inten) ' Importance:' num2str(importance) ' '];['After '...
        num2str(i) ' timesteps']});

    % subplot 1
    subplot(1,3,1);
    title('Initial Ground Structure');
    pcolor(myplain.realGround);
    shading interp;
    axis equal tight off ij;
```

```
    colormap(gray)
    freezeColors

    % subplot 2
    subplot(1,3,2);
    title('Evolving Trails');

    A=myplain.realGround;
    B=myplain.ground-myplain.initialGround;

    % shift B above the maximum of A
    B_shifted = B-min(B(:))+max(A(:))+1;

    % create fitted colormap out of two colormaps
    range_A = max(A(:))-min(A(:));
    range_B = max(B(:))-min(B(:));


    % to adjust caxis and colormap
    range_b = range_B;

    for i=1:10
       if  (range_b>=pathMax/10*(i-1))&&(range_b<=pathMax/10*i)
           range_B = pathMax/10*i;
       end
    end

    % adjusting colormap

    cm = [gray(ceil(64*range_A/range_B));flipud(summer(64))];

    % plotting
    pcolor(B_shifted)
    shading interp
    hold on
    contour(A)
    axis equal tight off ij;
    colormap(cm)
    caxis([min(A(:)) max(A(:))+range_B])

    freezeColors % http://www.mathworks.com/matlabcentral/fileexchange/7943

    % subplot 3
    subplot(1,3,3);
    title('Attractiveness');
    pcolor(vtr);
    shading interp;
    axis equal tight off ij;
    colormap(jet)
    freezeColors

```

```
    % plotting the pedestrians into the subplots
    for j=1:length(pedestrians)
        ped = pedestrians(j);

        subplot(1,3,1);
        title('Initial Ground Structure');

        hold on;
        plot(ped.position(2),ped.position(1),'wo');

        subplot(1,3,2);
        title('Evolving Trails');

        hold on;
        plot(ped.position(2),ped.position(1),'wo');

        subplot(1,3,3);
        title('Attractiveness');

        hold on;
        plot(ped.position(2),ped.position(1),'wo');
    end


    drawnow;
    C(i)=getframe(gcf);
end



% save data
savefile = sprintf('data/d%d_i%d_v%d_i%d_%s.mat',...
            dur, inten, vis, importance, site);
save(savefile, 'myplain', 'mysm');


%i = 1;
%str = sprintf('movie%d.avi',i);

%while(exist(str)>0)
%     i = i+1;
%     str = sprintf('movie%d.avi',i);
%end

%save movie to file
%movie2avi(C,str,'fps',3);

end
```

StateMachine.m

```
classdef StateMachine < handle
    % STATEMACHINE Handles the state changes in the simulation
    % Computes the change of the environment and moves all the pedestrians

    properties(SetAccess = public)
        plain;          % G ... the current plain
        pedestrians;    % Array of pedestrians which are currently walking
        importance;     % How to weight the vector to the destination
        entryPoints;    % entry points as specified by ginput
        paths;          % paths walked by pedestrians
        pathsSorted;    % paths sorted by what way they went
        speed;          % horizontal and vertical speed
        time;           % time in which the state machine is in

    end

    methods
        function obj = StateMachine(aPlain)
            % Constructor: set the plain
            obj.plain = aPlain;
        end

        function [Vtr] = transition(obj)
            % Does a transition in the state machine according to the plain
            % and the pedestrians.

            [n m] = size(obj.plain.ground);
            Vtr = zeros(n,m);

            % Make relative strength of path

            MakeRelativePath(obj.plain);

            % Generate new pedestrians and put it to the other
            % pedestrians

            newPed = Pedestrian(obj.entryPoints,obj.plain);
            obj.pedestrians = [obj.pedestrians,newPed];

            % Change the environment according to the pedestrian positions
            obj.plain.changeEnvironment(obj.pedestrians);

            % Compute the attractiveness for each point in the plain
            for i=1:n
                for j=1:m
                    Vtr(i,j) = obj.computeAttractiveness([i;j]);
                end
            end
```

```

            % Delete pedestrians which are at their destination or near
            ToDelete = false(1,length(obj.pedestrians));

            for i=1:length(obj.pedestrians)
                ToDelete(i) = isAtDestination(obj.pedestrians(i));
            end

            DeletedPed = obj.pedestrians(ToDelete);
            obj.pedestrians = obj.pedestrians(~ToDelete);

            % Save path of delted pedestrians and sort them
            for i=1:length(DeletedPed)
                % sort path
                newPath = Path(DeletedPed(i),obj.entryPoints,obj.speed,...
                    obj.plain,obj.time);
                obj.pathsSorted{newPath.type} = ...
                    [obj.pathsSorted{newPath.type}; ...
                    [newPath.type newPath.time newPath.timeOfArrival]];
                % save path to other paths
                obj.paths = [obj.paths, newPath];
            end

            % move and save way of pedestrians
            for i=1:length(obj.pedestrians)
                movePedestrian(obj,i,Vtr);
                saveWay(obj.pedestrians(i),obj.plain);
            end

        end


        function movePedestrian(obj,pedestNum,vtr)
            % Moves a pedestrian according to the attractiveness of the
            % neighbourhood and its destination

            pedest = obj.pedestrians(pedestNum);

            maxvtr = -inf;
            maxcoords = [0;0];

            % compute the maximum value of vtr in the neighbourhood and
            % save the direction to it
            for i = -1:1
                for j = -1:1
                    y = pedest.position(1)+i;
                    x = pedest.position(2)+j;
                    if(obj.plain.isPointInPlain(y,x))
                        if maxvtr < vtr(y,x)
                            maxvtr = vtr(y,x);
                            maxcoords = [i j];
```

```
                    end
                end

            end
        end

        % normalize the gradient vector (but check for zero division)
        if(norm(maxcoords)>0)
            maxcoords = maxcoords / norm(maxcoords);
        end

        % compute the vector to the destination and normalize it
        toDest = pedest.destination - pedest.position;
        toDest = toDest ./ norm(toDest);

        % add both vectors, but multiply the toDest vector with
        % importance to get better results
        moveDir = obj.importance * toDest + maxcoords;

        % compute the angle of the directional vector
        alpha = atan(moveDir(1)/moveDir(2));

        % Because tan is pi periodic we have to add pi to the angle
        % if x is less than zero
        if moveDir(2) < 0
            alpha = alpha + pi;
        end

        % Define the direction vectors
        up = [-1 0];
        down = [1 0];
        left = [0 -1];
        right = [0 1];

        % Initialize the move vector
        move = [0 0];

        % Shortcut for pi/8
        piEi = pi/8;

        % Check the angle of the resulting vector and choose
        % the moving direction accordingly

        if (alpha < -3*piEi) || (alpha >  11*piEi)
            % move up
            move = up;

        elseif (alpha >= -3*piEi) && (alpha < -piEi)
            % move right up
            move = up + right;

```

```
        elseif (alpha >= -piEi) && (alpha < piEi)
            % move right
            move = right;

        elseif (alpha >= piEi) && (alpha < 3*piEi)
            % move down right
            move = down + right;

        elseif (alpha >= 3*piEi) && (alpha < 5*piEi)
            % move down
            move = down;

        elseif (alpha >= 5*piEi) && (alpha < 7*piEi)
            % move down left
            move = down + left;

        elseif (alpha >= 7*piEi) && (alpha < 9*piEi)
            % move left
            move = left;

        elseif (alpha >= 9*piEi) && (alpha < 11*piEi)
            % move up left
            move = up + left;
        end

        % Actually move the pedestrian
        pedest.position = pedest.position + move;
    end


    function Vtr = computeAttractiveness(obj,coords)
        % This function computes the sum of all attracivenesses
        % of the whole area from the viewpoint of coords

        % Get the visibility at point coords
        visibility = obj.plain.visibility(coords(1),coords(2));

        % Get the current ground structure
        G = obj.plain.ground;
        [n m] = size(G);

        % Efficient implementation for the sum
        [A,B]=meshgrid(((1:m)-coords(2)).^2,((1:n)-coords(1)).^2);
        S=-sqrt(A+B);
        S = exp(S/visibility);
        S = S.*G;
        Vtr = sum(sum(S));

        % Average the sum over the number of squares in the plain
        Vtr = Vtr/(m*n);
    end
```

```
    end

end
```

visualization.m

```
function visualization

global f1;
f1 = figure('OuterPosition',[0 0 700 600]);

files = dir('data/*stm.mat');

for file=1:numel(files)

    clear myplain mysm;
    clf;

    filename = files(file).name;

    % extract parameter values
    params = sscanf(filename, 'd%d_i%d_v%f_i%f_.mat');
    dur = params(1);
    inten = params(2);
    vis = params(3);
    importance = params(4);
    location = 'fri';

    plot_paths(filename);
    [n t dist dist_std traveltime] = completed_paths(filename);

    data = sprintf('%i,%i,%f,%f,%i,%f,%f,%f,%f', ...
            dur,inten,vis,importance,n,t,dist,dist_std,traveltime);
    disp(data);

end

end



%------------------------------------------------------
function [n t_norm dist dist_std traveltime] = completed_paths(filename)
    load(strcat('data/',filename));

    % number of paths completed
```

```
    n = length(mysm.paths);

    % travel time
    t = zeros(3,n);

    for i=1:n
        path=mysm.paths(i);
        start = path.coordinates(1,:);
        dest = path.coordinates(end,:);
        dist = sqrt(sum((dest-start).^2));

        traveltime = path.time;

        t(:,i) = [dist;traveltime;dist/traveltime];
    end

    t_norm = mean(t(3,:));
    dist = mean(t(1,:));
    dist_std = std(t(1,:));
    traveltime = mean(t(2,:));

end



%----------------------------------------------------
function plot_paths(filename)
    global f1;
    load(strcat('data/',filename));


    % plot ground structure
    subplot(1,2,1);
    pathMax = 100;
    title('Evolving Trails');

    A=myplain.realGround;
    B=myplain.ground-myplain.initialGround;

    % shift B above the maximum of A
    B_shifted = B-min(B(:))+max(A(:))+1;

    % create fitted colormap out of two colormaps
    range_A = max(A(:))-min(A(:));
    range_B = max(B(:))-min(B(:));


    % to adjust caxis and colormap
    range_b = range_B;

    for i=1:10
```

```
        if  (range_b>=pathMax/10*(i-1))&&(range_b<=pathMax/10*i)
            range_B = pathMax/10*i;
        end
    end

    % adjusting colormap

    cm = [gray(ceil(64*range_A/range_B));flipud(summer(64))];

    % plotting
    pcolor(B_shifted)
    shading interp
    hold on
    contour(A)
    axis equal tight off ij;
    colormap(cm)
    caxis([min(A(:)) max(A(:))+range_B])

    freezeColors % http://www.mathworks.com/matlabcentral/fileexchange/7943


    % plot vector paths
    for ped=mysm.pedestrians
        plot(ped.way(:,1), ped.way(:,2));
    end

    % plot cities
    if(strfind(filename,'fri'))
        p = [34 26;30 27;35 31;30 12;31 23;28 21;13 12;34 20;21 33;26 ...
            35;27 26;21 19;21 29;14 34];
    else
        p = [23 31;29 23;34 32;27 28;17 40;32 19];
    end
    plot(p(:,1),p(:,2),'wo');


    hold off;



    % plot travel time
    subplot(1,2,2);
    hold on;
    for path=mysm.paths
        plot(path.timeOfArrival,path.time,'o');
        text(path.timeOfArrival,path.time,sprintf('%i',path.type));
    end

    axis square;
    hold off;

```

```

    saveas(f1,strcat('images2/',filename,'.png'));

end
```