

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

1	Introduction	1
2	Description of the Model	2
2.1	Ground Structure	3
2.2	Attractiveness of Trail Segment	3
2.3	Pedestrians Walking Direction	4
3	Description of the Path-Evaluation Function	4
4	Implementation	5
4.1	Interface for real Elevation Data	5
4.2	Implementation of the Path-Evaluation Function	6
5	Simulation Results and Discussion	6
5.1	Parameter variation	7
5.2	Existing road structure	9
6	Summary and Outlook	9
7	References	9
8	Appendix: Matlab Code	11

Abstract

Individual contributions

Code was developed in team work. Christoph focused on the model itself, while Benjamin focused on visualization of results and building test cases. Appropriately, for the report, Christoph focused on the description and implementation of the model, while Benjamin focused on simulation results.

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

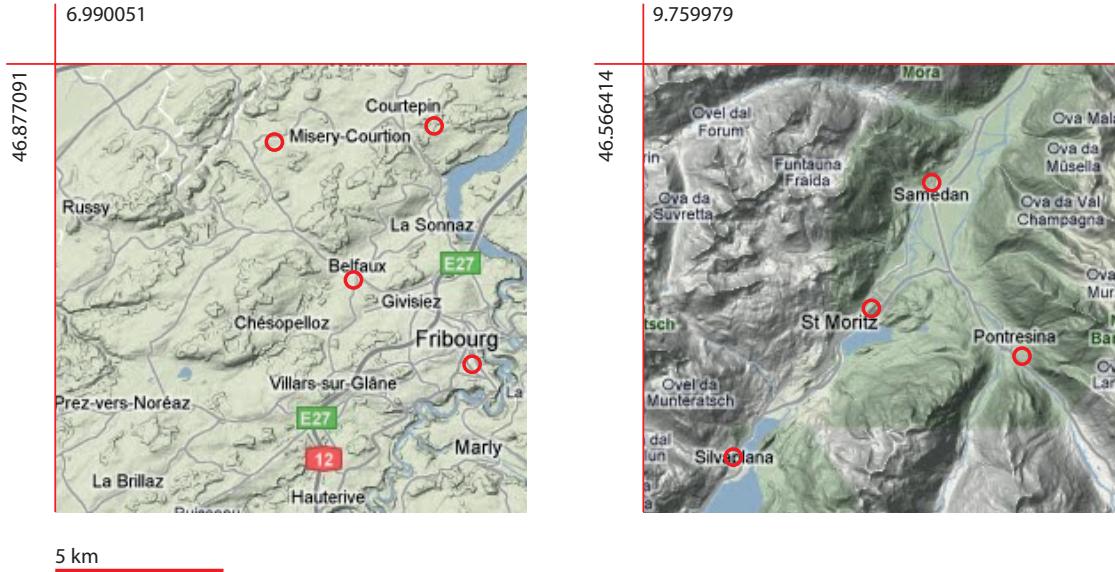


Figure 1. The two study sites: a relatively flat region around Fribourg and a more mountainous region around St. Moritz. Coordinates are given as latitude/longitude in WGS84.

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.

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 attractiveness

of a trail segment 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 a certain point of the plain it changes the comfort of walking there. This is because pedestrians trample down the vegetation. This is described by

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where $\frac{G(r_\alpha, t) - G_0(r_\alpha, t)}{G_{max}(r_\alpha, t) - G_0(r_\alpha, t)}$ is the relative ground.

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 ($5 \times 5 \text{ km}$ for each site) came in a resolution of $25 \times 25 \text{ m}$. To use this elevation data in our model first the resolution had to be lowered to $500 \times 500 \text{ m}$ for computational reasons and then the data had to be adjusted to use it as initial comfort of walking G_0 .

As high values of comfort of walking represent paths which are favored for walking on and high values in the elevation data are mountains which are not favored for walking on the elevation model had to be inverted. So in the inverted elevation model valleys became mountains and mountains became valleys. Therefore the initial comfort of walking was set to

$$G_0 = \arg\max(E) - E \quad (10)$$

where E is the original elevation model.

4.2 Implementation of the Path-Evaluation Function

To add the path-evaluation function to the model the pedestrian class had to be changed and a new class, the path class was created. First the changes to the pedestrian class are described followed by the introduction of the path class.

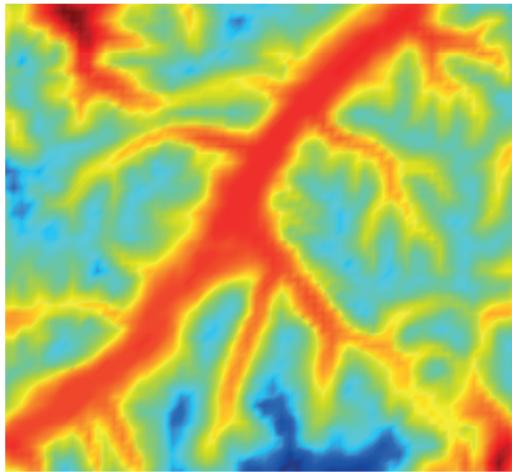
To evaluate the travel-time equation (Eq. 8) the relative ground $\frac{G(r_{\alpha,t}) - G_0(r_{\alpha,t})}{G_{\max}(r_{\alpha,t}) - G_0(r_{\alpha,t})}$ and the coordinates of the walked way by the pedestrian have to be known. Therefore the pedestrian class was changed to save the coordinates and the relative ground on this coordinates on every time step of the state machine.

Furthermore on every time step when a pedestrian arrives at his destination the information about the walked path is used to create an object of the new path class. An object of the path class has the following properties: coordinates, relative ground on coordinates, time, time of arrival and type. The coordinates and the relative ground on the coordinates are provided by the pedestrian object itself while time is the result of the travel-time equation and time of arrival the time step in which the state machine is in when the pedestrian arrives. The type of the path is a number which was assigned to the path depending on which way (which start and which destination) the pedestrian went. All the created path objects are then saved in the simulation for later analysis.

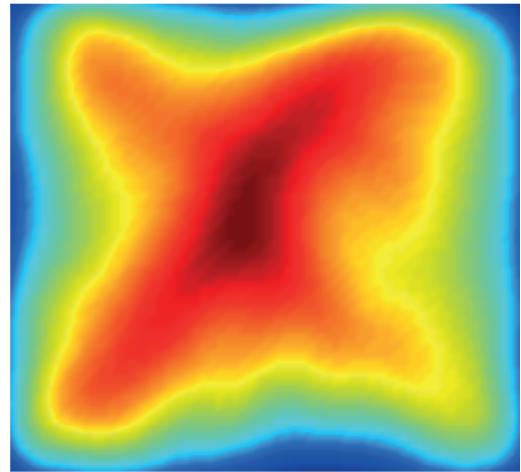
5 Simulation Results and Discussion

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 parameters (they are dependent on model specifics such as the model scale). Since our initial hypotheses cannot be conclusively addressed without dealing with the issue of model parameters, we first discuss the effects parameter variation has on model results. We then compare our results with existing road structures.

Attractiveness



Visibility = 0.3



Visibility = 4

Figure 2. The effect of the visibility parameter on attractiveness structure. At low visibilities, the attractiveness reflects the terrain, whereas at high visibilities agents focus more on their destination and terrain details are lost.

5.1 Parameter variation

A systematic parameter variation study was performed. The two test sites were set up with fixed destinations corresponding to the largest cities in the area. An appropriate range of possible values was chosen for each of the four independent model parameters and the model run for 100 steps for each possible parameter combination.

Visibility

Unfortunately, due to a MATLAB error, the visibility parameter could not be systematically analyzed. The following discussion is therefore of a qualitative nature. As described in subsection 2.2, visibility controls how fast the influence of an agent's surroundings decreases with distance. As can be seen in Figure 2, a higher visibility causes the attractiveness to "soften": the terrain details present at a visibility of 0.3 are lost at a visibility of 4. This causes the agent to consider a larger area in his decision, resulting in a more efficient path taken. Also, paths resulting paths may be more diverse, as different paths have similar efficiency when viewed over a large area rather than simply in the immediate vicinity.

Importance

As described in subsection 2.3, the importance parameter controls how the destination is weighted against an agent's preference for walking in the most attractive direction. A higher importance causes the destination vector to be weighed more heavily, whereas a lower importance causes the attractiveness vector to be weighed more heavily. In Table 2, the average number of completed paths (agents having reached their destination) after 100 model steps is listed. As might be expected, a higher importance results in significantly more agents reaching their destination. If the importance is lower, agents wander around in the direction where walking is easiest, never reaching their destination (Figure 3). It is difficult, however, to determine what value of importance is the "correct" one. Real agents may prefer taking a detour if it means they can walk more comfortably. Also, Table 3 indicates that *when* agents manage to reach their destination, their travel times don't vary considerably. However, all-in-all, since considerably more agents manage to reach their destination within the timespan of 100 model steps when importance is high, this can be considered the more efficient system.

Table 2. The average number of completed paths (agents having reached their destination) after 100 model steps as a function of the parameter *importance*.

<i>importance</i>	0.5	1	1.6
<i>Fribourg</i>	3.2	23.2	88.2
<i>St. Moritz</i>	5.9	26.4	89.3

Table 3. The average normed travel time (time divided by point-to-point distance) of completed paths after 100 model steps as a function of the parameter *importance*.

<i>importance</i>	0.5	1	1.6
<i>Fribourg</i>	0.15	0.22	0.16
<i>St. Moritz</i>	0.16	0.18	0.20

intensity

Durability

The parameter *durability*

higher clustering, but only at high intensities. more pronounced at fribourg, since st. moritz is constrained anyway.

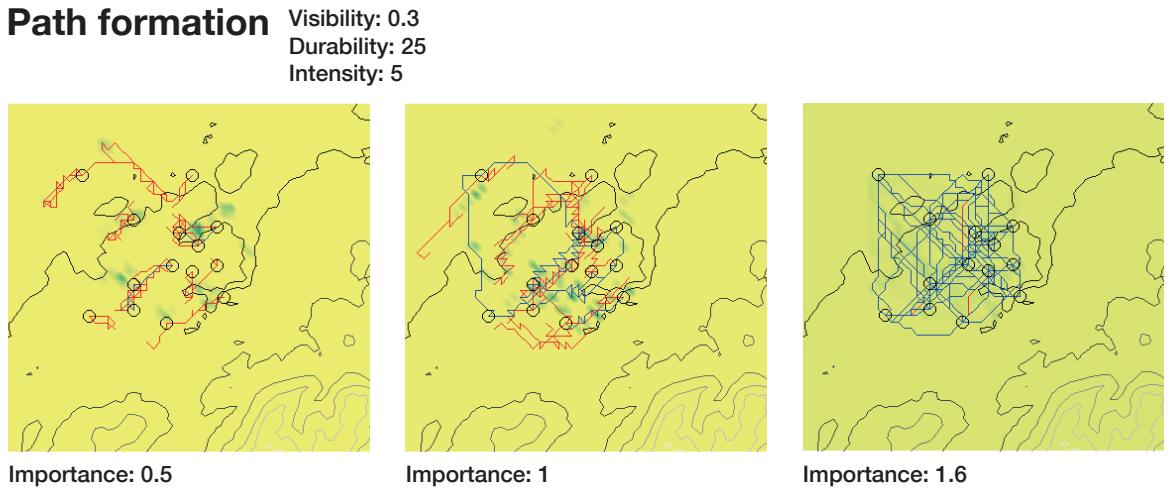


Figure 3. Path formation as a function of parameter *importance* at site Fribourg. Blue lines are completed paths, red lines are agents that have not yet reached their destination. The number of agents that reach their destination increases with increasing importance: whereas at an importance of 1.6 many paths are completed, at an importance of 0.5 most agents fail to reach their destination, preferring to wander aimlessly.

5.2 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

- (2007). *Vector25: Das digitale Landschaftsmodell der Schweiz*. Bundesamt für Landestopografie.
- Helbing, D., Keltsch, J., & Molnár, P. (1997). Modelling the evolution of human trail systems. *Nature*, 388(3).

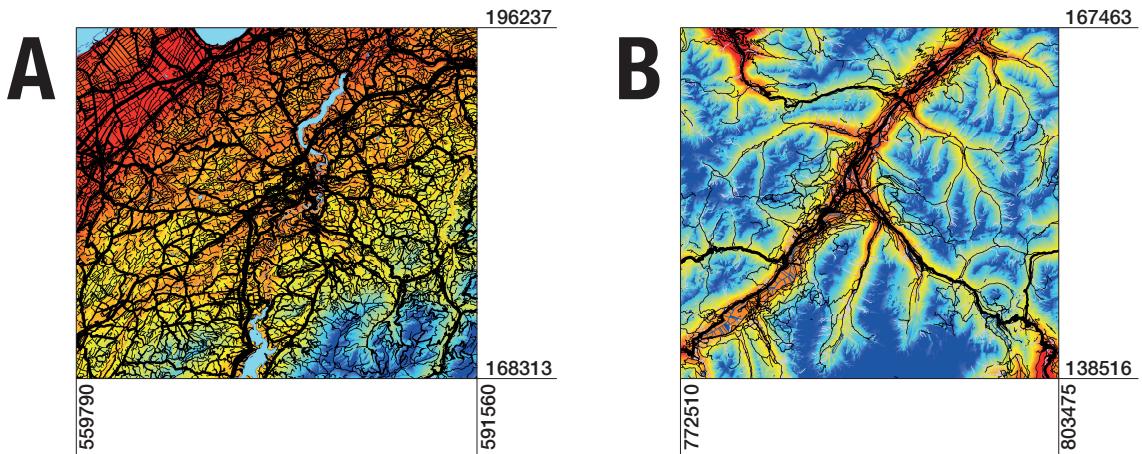


Figure 4. The road structure at the two study sites (A: Fribourg, B: St. Moritz). Coordinates are given in the reference frame CH1903. The line weight is derived from road class order (highways and 1. order roads are heavier in weight than higher order roads; see vec (2007)) and does not necessarily correlate to throughput.

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

```
1 % location of cities on original grid (25x25m)
2 stm_p = [454 620; 567 452; 674 638; 528 542; 328 785; 623 372];
3 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];
4
5 % location of cities on reduced grid (500x500m)
6 stm_p = [23 31;29 23;34 32;27 28;17 40;32 19];
7 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];
8
9
10
11 %dur = 25; % Durability
12 %inten = 10; % Intensity
13 %vis = 0.3, 1, 4; % Visability
14 %importance = 1.6;
15 %location = 'fri'
16 %numframes = 100
17
18 for vis=[0.3 1 4]
19     for dur = [5 25 50]
20         for inten = [5 10 30]
21             for importance = [0.5 1 1.6]
22                 smDriver(dur, inten, vis, importance, 'fri', 100);
23                 smDriver(dur, inten, vis, importance, 'stm', 100);
24             end
25         end
26     end
27 end
```

Path.m

```
1 classdef Path < handle
2     %PATH our path class
3
4     properties(SetAccess = public)
5         coordinates;
6         type;
7         time;
8         timeOfArrival;
9         relativeGround;
10    end
```

```

11
12     methods
13
14         function obj = Path(ped,entryP,speed,aPlain,aTime)
15             % takes deleted pedestrian and generates its path with
16             % properties
17             obj.coordinates = [ped.way; ped.destination];
18             obj.relativeGround = [ped.relativeGround; ...
19                 aPlain.relativePath(ped.destination(1),ped.destination(2))];
20             obj.timeOfArrival = aTime;
21             PathType(obj,entryP);
22             PathTime(obj,speed,aPlain);
23     end
24
25     function PathType(obj,entryP)
26         % checks which from the possible paths the pedestrian went
27
28         % generate possible paths
29         PossPathIndex = nchoosek(1:length(entryP),2);
30
31         PossPath.origin = zeros(size(PossPathIndex));
32         PossPath.destination = zeros(size(PossPathIndex));
33
34         for i=1:size(PossPathIndex,1)
35             PossPath.origin(i,:) = entryP(PossPathIndex(i,1),:);
36             PossPath.destination(i,:) = entryP(PossPathIndex(i,2),:);
37         end
38
39         % check which from the possible paths the path is
40         for i=1:size(PossPathIndex,1)
41
42             if ((obj.coordinates(1,:)==PossPath.origin(i,:)|...
43                 obj.coordinates(1,:)==PossPath.destination(i,:))&...
44                 (obj.coordinates(end,:)==PossPath.origin(i,:)|...
45                 obj.coordinates(end,:)==PossPath.destination(i,:)))
46                 obj.type=i;
47             end
48
49         end
50     end
51
52     function PathTime(obj,speed,aPlain)
53         % calculates the time it takes the pedestrian to walk the path
54
55         path.horiz = obj.coordinates;
56
57         % extracting height info from elevation model
58         path.vert = zeros(size(path.horiz,1),1);
59         for i=1:size(path.horiz,1)
60             path.vert(i) = ...
61                 aPlain.realGround(path.horiz(i,1),path.horiz(i,2));

```

```

61      end
62
63
64
65      % calculating horizontal and vertical distance from path
66      % (vertical distance only taken if path goes uphill)
67
68      dist_horiz = zeros(size(path.horiz,1)-1,1);
69      dist_vert = zeros(size(path.horiz,1)-1,1);
70      relativeGroundMean = zeros(size(path.horiz,1)-1,1);
71
72      for i=1:size(path.horiz,1)-1
73
74          % calculate horizontal distance
75          delta_horiz = norm(path.horiz(i+1,:)-path.horiz(i,:));
76          dist_horiz(i,1) = delta_horiz;
77
78          % calculate horizontal distance
79          delta_vert = path.vert(i+1)-path.vert(i);
80          if delta_vert>=0
81              dist_vert(i,1) = delta_vert;
82          else
83              dist_vert(i,1) = 0;
84          end
85
86          % calculate relative ground between the gridpoints
87          relativeGroundMean(i,1) = ...
88              (obj.relativeGround(i+1,1)-obj.relativeGround(i,1))/2;
89
90      end
91
92      % scale horizontal distance with grid size and calculate walking
93      % time
94
95      speed.horizontal.real = speed.horizontal.min + ...
96          (speed.horizontal.max-speed.horizontal.min)*relativeGroundMean;
97
98      obj.time = ...
99          sum(aPlain.gridSize*dist_horiz./speed.horizontal.real + ...
100             dist_vert/speed.vertical);
101
102 end

```

Pedestrian.m

```

1 classdef Pedestrian < handle
2     %PEDESTRIAN our pedestrian class
3
4     properties(SetAccess = private )
5         destination;
6         way;
7         relativeGround;
8     end
9
10    properties(SetAccess = public)
11        position;
12    end
13
14    methods
15        function obj = Pedestrian(entryP,aPlain)
16            % generate new pedestrian and randomly choose origin and
17            % destination from given entry points
18
19            r = randperm(length(entryP));
20            orig = entryP(r(1),:);
21            dest = entryP(r(2),:);
22            obj.way = orig;
23            obj.destination = dest;
24            obj.position = orig;
25            obj.relativeGround = aPlain.relativePath(obj.position(1),...
26                obj.position(2));
27        end
28
29        function set.position(obj,pos)
30            % put pedestrian to position
31            obj.position = pos;
32        end
33
34        function saveWay(obj,aPlain)
35            % save way of pedestrian and relative strength of ground
36            obj.way = [obj.way; obj.position];
37            obj.relativeGround = [obj.relativeGround;...
38                aPlain.relativePath(obj.position(1),obj.position(2))];
39        end
40
41        function val = isAtDestination(obj)
42            % check if pedestrian arrived at destination
43            val = (norm(obj.position - obj.destination)<2);
44        end
45    end
46 end

```

Plain.m

```

1 classdef Plain < handle
2     %PLAIN Saves state of the plain
3
4     properties(SetAccess = public)
5         ground;          % The current ground structure
6         initialGround; % initial ground model
7         groundMax;      % The maximum values of the walking comfort
8         intensity;       % The footprint intensity
9         durability;     % The durability of trails
10        visibility;    % The visibility at each point
11        realGround;    % real, not inverted, ground
12        gridSize;      % grid size of plain in m
13        relativePath;  % relative strength of path
14    end
15
16    methods
17        function obj=Plain(aInitialGround,aGroundMax,aIntensity, ...
18                           aDurability,aVisibility,aRealGround,aGridSize)
19            initSize = size(aInitialGround);
20            % Check if initialGround has same size as intensity and
21            % durability matrix
22
23            if((nnz(initSize == size(aIntensity)) == 2) &&...
24                (nnz(initSize == size(aDurability))==2) &&...
25                (nnz(initSize == size(aGroundMax))==2) &&...
26                (nnz(initSize == size(aVisibility))==2) &&...
27                (nnz(initSize == size(aRealGround))==2))
28
29                obj.ground = aInitialGround;
30                obj.groundMax = aGroundMax;
31                obj.initialGround = aInitialGround;
32                obj.intensity = aIntensity;
33                obj.durability = aDurability;
34                obj.visibility = aVisibility;
35                obj.realGround = aRealGround;
36                obj.gridSize = aGridSize;
37            else
38                error('PLAIN(): initialGround must be same size as ...
39                      intensity and durability');
40            end
41        end
42        function changeEnvironment(obj,pedestrians)
43            % Changes the environment according to the positions of the
44            % pedestrians
45            [n m] = size(obj.ground);
46            pedAt = sparse(n,m);
47
48            for i=1:length(pedestrians)
49                ped = pedestrians(i);

```

```

50         pedAt(ped.position(1),ped.position(2)) = ...
51             pedAt(ped.position(1),ped.position(2)) + 1;
52     end
53
54     % Change the environment on each square of the plain
55     for i=1:n
56         for j=1:m
57             % Change the ground according to the formula
58             obj.ground(i,j) = obj.ground(i,j) + ...
59                 1/obj.durability(i,j) * (obj.initialGround(i,j)-...
60                 obj.ground(i,j)) + obj.intensity(i,j) * ...
61                 (1-(obj.ground(i,j)/obj.groundMax(i,j))) * ...
62                 pedAt(i,j);
63
64             % Check for the boundaries of the ground values
65             if(obj.ground(i,j) > obj.groundMax(i,j))
66                 obj.ground(i,j) = obj.groundMax(i,j);
67             elseif(obj.ground(i,j) < obj.initialGround(i,j))
68                 obj.ground(i,j) = obj.initialGround(i,j);
69             end
70         end
71     end
72
73     end
74
75     function val = isPointInPlain(obj,y,x)
76         % Returns wheter or not a point (x,y) is in this plain
77         val = (y>0 && x>0);
78         val = val && y<=size(obj.ground,1) && x<=size(obj.ground,2);
79
80     end
81
82     function MakeRelativePath(obj)
83         % Calculates relative path strength (1 = maximal path)
84         obj.relativePath = (obj.ground-obj.initialGround)./...
85             (obj.groundMax-obj.initialGround);
86     end
87 end
88
89 end

```

smDriver.m

```

1 function [mysm,vtr] = smDriver(dur, inten, vis, importance, site, numframes)
2 %SMDRIVER Sets up a simulation
3
4 f1 = figure('OuterPosition',[0 0 700 600]);
5 winsize = get(f1,'Position');

```

```

6 %numframes = 200;
7
8
9 % selecting elevation model
10 load elevation
11 if(strcmp(site,'stm'))
12     elevation = stmoritz(1:1140,:); % for St. Moritz
13     entryPoints = [23 31;29 23;34 32;27 28;17 40;32 19];
14 elseif(strcmp(site,'fri'))
15     elevation = friburg(1:1100,1:1260); % for Friburg
16     entryPoints = [34 26;30 27;35 31;30 12;31 23;28 21;13 12;34 20;21 ...
17         33;26 35;27 26;21 19;21 29;14 34];
18 end
19
20 % resizing elevation model (original elevation dim must be multiple of 20)
21 elevation_re = zeros(size(elevation)/20);
22 [m n] = size(elevation);
23
24 for i=20:20:m
25     for j=20:20:n
26         elevation_re(i/20,j/20) = mean2(elevation(i-19:i,j-19:j));
27     end
28 end
29
30 [m n] = size(elevation_re);
31
32
33 % Set the parameters
34 %dur = 25;                      % Durability
35 %inten = 10;                     % Intensity
36 %vis = 4;                        % Visibility
37 %importance = 1.6;               % Weight of the destination vector
38 speed.horizontal.min = 4000;      % min horizontal speed in m/h
39 speed.horizontal.max = 6000;      % max horizontal speed in m/h
40 speed.vertical = 500;            % vertical speed in m/h
41 gridSize = 500;                 % grid size of plain in m
42 pathMax = 100;                  % maximal value of a path
43
44
45 initialGround = max(max(elevation_re))-elevation_re; % inverting elevation
46 groundMax = initialGround + ones(m,n)*pathMax;
47 intensity = ones(m,n) * inten;
48 durability = ones(m,n) * dur;
49 visibility = ones(m,n) * vis;
50 elevation = elevation_re;
51
52 % create new plain with the specified values
53 myplain = Plain(initialGround,groundMax,intensity,durability,visibility, ...
54     elevation,gridSize);
55

```

```

56 % show the plain for input of the entry points
57 %pcolor(myplain.realGround);
58 %colormap(gray);
59 %shading interp;
60 %axis ij;
61 %entryPoints = ginput;
62 %entryPoints = floor([entryPoints(:,2) entryPoints(:,1)]);
63
64 % create a state machine with the specified plain
65 mysm = StateMachine(myplain);
66 mysm.importance = importance;
67 mysm.entryPoints = entryPoints;
68 mysm.speed = speed;
69 % make cell, where possible paths on Plain are later saved
70 noPossPaths = length(nchoosek(1:length(mysm.entryPoints), 2));
71 mysm.pathsSorted = cell(1,noPossPaths);
72
73 % Do 'numframes' timesteps
74 C(1) = getframe(gcf);
75 for i=1:numframes
76
77 % print every 20th timestep into a .png file
78 if(mod(i,20)==0 && i >0)
79     str = sprintf('images/im_%d_d%d_i%d_v%d_%d.png',...
80                 importance,dur,inten,vis,i);
81     saveas(f1,str);
82 end
83
84 % compute a new transition in the state machine
85 vtr = mysm.transition;
86
87 % tell the StateMachine in which state it is in
88 mysm.time = i;
89
90 % display how many pedestrians are on the plane
91 pedestrians = mysm.pedestrians;
92 fprintf('Number of pedestrians: %d\n',length(pedestrians));
93
94 % making the 3 subplots
95 clf(f1);
96 suptitle({[];[];['Grid:' num2str(m) 'x' num2str(n)];['Durability:' ...
97             num2str(dur) ' Visibility:' num2str(vis) ' '];[ 'Intensity:' ...
98             num2str(inten) ' Importance:' num2str(importance) ' '];['After '...
99             num2str(i) ' timesteps']} );
100
101 % subplot 1
102 subplot(1,3,1);
103 title('Initial Ground Structure');
104 pcolor(myplain.realGround);
105 shading interp;
106 axis equal tight off ij;

```

```

107 colormap(gray)
108 freezeColors
109
110 % subplot 2
111 subplot(1,3,2);
112 title('Evolving Trails');
113
114 A=myplain.realGround;
115 B=myplain.ground-myplain.initialGround;
116
117 % shift B above the maximum of A
118 B_shifted = B-min(B(:))+max(A(:))+1;
119
120 % create fitted colormap out of two colormaps
121 range_A = max(A(:))-min(A(:));
122 range_B = max(B(:))-min(B(:));
123
124
125 % to adjust caxis and colormap
126 range_b = range_B;
127
128 for i=1:10
129     if (range_b>=pathMax/10*(i-1)) && (range_b<=pathMax/10*i)
130         range_B = pathMax/10*i;
131     end
132 end
133
134 % adjusting colormap
135
136 cm = [gray(ceil(64*range_A/range_B));flipud(summer(64))];
137
138 % plotting
139 pcolor(B_shifted)
140 shading interp
141 hold on
142 contour(A)
143 axis equal tight off ij;
144 colormap(cm)
145 caxis([min(A(:)) max(A(:))+range_B])
146
147 freezeColors % http://www.mathworks.com/matlabcentral/fileexchange/7943
148
149 % subplot 3
150 subplot(1,3,3);
151 title('Attractiveness');
152 pcolor(vtr);
153 shading interp;
154 axis equal tight off ij;
155 colormap(jet)
156 freezeColors
157

```

```

158 % plotting the pedestrians into the subplots
159 for j=1:length(peDESTRIANS)
160     ped = peDESTRIANS(j);
161
162     subplot(1,3,1);
163     title('Initial Ground Structure');
164
165     hold on;
166     plot(ped.position(2),ped.position(1), 'wo');
167
168     subplot(1,3,2);
169     title('Evolving Trails');
170
171     hold on;
172     plot(ped.position(2),ped.position(1), 'wo');
173
174     subplot(1,3,3);
175     title('Attractiveness');
176
177     hold on;
178     plot(ped.position(2),ped.position(1), 'wo');
179 end
180
181
182 drawnow;
183 C(i)=getframe(gcf);
184 end
185
186
187
188 % save data
189 %savefile = sprintf('data/d%d_i%d_v%d_i%d_%s.mat',...
190 %                 dur, inten, vis, importance, site);
191 %save(savefile, 'myplain', 'mysm');
192
193
194 %i = 1;
195 %str = sprintf('movie%d.avi',i);
196
197 %while(exist(str)>0)
198 %    i = i+1;
199 %    str = sprintf('movie%d.avi',i);
200 %end
201
202 %save movie to file
203 %movie2avi(C,str,'fps',3);
204
205 end

```

StateMachine.m

```
1 classdef StateMachine < handle
2     % STATEMACHINE Handles the state changes in the simulation
3     % Computes the change of the environment and moves all the pedestrians
4
5 properties(SetAccess = public)
6     plain;          % G ... the current plain
7     pedestrians;    % Array of pedestrians which are currently walking
8     importance;     % How to weight the vector to the destination
9     entryPoints;    % entry points as specified by ginput
10    paths;          % paths walked by pedestrians
11    pathsSorted;    % paths sorted by what way they went
12    speed;          % horizontal and vertical speed
13    time;           % time in which the state machine is in
14
15 end
16
17 methods
18     function obj = StateMachine(aPlain)
19         % Constructor: set the plain
20         obj.plain = aPlain;
21     end
22
23     function [Vtr] = transition(obj)
24         % Does a transition in the state machine according to the plain
25         % and the pedestrians.
26
27         [n m] = size(obj.plain.ground);
28         Vtr = zeros(n,m);
29
30         % Make relative strength of path
31
32         MakeRelativePath(obj.plain);
33
34         % Generate new pedestrians and put it to the other
35         % pedestrians
36
37         newPed = Pedestrian(obj.entryPoints,obj.plain);
38         obj.pedestrians = [obj.pedestrians,newPed];
39
40         % Change the environment according to the pedestrian positions
41         obj.plain.changeEnvironment(obj.pedestrians);
42
43         % Compute the attractiveness for each point in the plain
44         for i=1:n
45             for j=1:m
46                 Vtr(i,j) = obj.computeAttractiveness([i;j]);
47             end
48         end
```

```

49
50      % Delete pedestrians which are at their destination or near
51     ToDelete = false(1,length(obj.pedestrians));
52
53      for i=1:length(obj.pedestrians)
54         ToDelete(i) = isAtDestination(obj.pedestrians(i));
55      end
56
57      DeletedPed = obj.pedestriansToDelete;
58      obj.pedestrians = obj.pedestrians(~ToDelete);
59
60      % Save path of deleted pedestrians and sort them
61      for i=1:length(DeletedPed)
62          % sort path
63          newPath = Path(DeletedPed(i),obj.entryPoints,obj.speed, ...
64              obj.plain,obj.time);
65          obj.pathsSorted{newPath.type} = ...
66              [obj.pathsSorted{newPath.type}; ...
67                  newPath.type newPath.time newPath.timeOfArrival]];
68          % save path to other paths
69          obj.paths = [obj.paths, newPath];
70      end
71
72      % move and save way of pedestrians
73      for i=1:length(obj.pedestrians)
74          movePedestrian(obj,i,Vtr);
75          saveWay(obj.pedestrians(i),obj.plain);
76      end
77
78  end
79
80
81  function movePedestrian(obj,pedestNum,vtr)
82      % Moves a pedestrian according to the attractiveness of the
83      % neighbourhood and its destination
84
85      pedest = obj.pedestrians(pedestNum);
86
87      maxvtr = -inf;
88      maxcoords = [0;0];
89
90      % compute the maximum value of vtr in the neighbourhood and
91      % save the direction to it
92      for i = -1:1
93          for j = -1:1
94              y = pedest.position(1)+i;
95              x = pedest.position(2)+j;
96              if(obj.plain.isPointInPlain(y,x))
97                  if maxvtr < vtr(y,x)
98                      maxvtr = vtr(y,x);
99                      maxcoords = [i j];

```

```

100           end
101       end
102
103   end
104 end
105
106 % normalize the gradient vector (but check for zero division)
107 if(norm(maxcoords)>0)
108     maxcoords = maxcoords / norm(maxcoords);
109 end
110
111 % compute the vector to the destination and normalize it
112 toDest = pedest.destination - pedest.position;
113 toDest = toDest ./ norm(toDest);
114
115 % add both vectors, but multiply the toDest vector with
116 % importance to get better results
117 moveDir = obj.importance * toDest + maxcoords;
118
119 % compute the angle of the directional vector
120 alpha = atan(moveDir(1)/moveDir(2));
121
122 % Because tan is pi periodic we have to add pi to the angle
123 % if x is less than zero
124 if moveDir(2) < 0
125     alpha = alpha + pi;
126 end
127
128 % Define the direction vectors
129 up = [-1 0];
130 down = [1 0];
131 left = [0 -1];
132 right = [0 1];
133
134 % Initialize the move vector
135 move = [0 0];
136
137 % Shortcut for pi/8
138 piEi = pi/8;
139
140 % Check the angle of the resulting vector and choose
141 % the moving direction accordingly
142
143 if (alpha < -3*piEi) || (alpha > 11*piEi)
144     % move up
145     move = up;
146
147 elseif (alpha >= -3*piEi) && (alpha < -piEi)
148     % move right up
149     move = up + right;
150

```

```

151     elseif (alpha >= -piEi) && (alpha < piEi)
152         % move right
153         move = right;
154
155     elseif (alpha >= piEi) && (alpha < 3*piEi)
156         % move down right
157         move = down + right;
158
159     elseif (alpha >= 3*piEi) && (alpha < 5*piEi)
160         % move down
161         move = down;
162
163     elseif (alpha >= 5*piEi) && (alpha < 7*piEi)
164         % move down left
165         move = down + left;
166
167     elseif (alpha >= 7*piEi) && (alpha < 9*piEi)
168         % move left
169         move = left;
170
171     elseif (alpha >= 9*piEi) && (alpha < 11*piEi)
172         % move up left
173         move = up + left;
174     end
175
176     % Actually move the pedestrian
177     pedest.position = pedest.position + move;
178 end
179
180
181 function Vtr = computeAttractiveness(obj,coords)
182     % This function computes the sum of all attractivenesses
183     % of the whole area from the viewpoint of coords
184
185     % Get the visibility at point coords
186     visibility = obj.plain.visibility(coords(1),coords(2));
187
188     % Get the current ground structure
189     G = obj.plain.ground;
190     [n m] = size(G);
191
192     % Efficient implementation for the sum
193     [A,B]=meshgrid((1:m)-coords(2)).^2, ((1:n)-coords(1)).^2;
194     S=-sqrt(A+B);
195     S = exp(S/visibility);
196     S = S.*G;
197     Vtr = sum(sum(S));
198
199     % Average the sum over the number of squares in the plain
200     Vtr = Vtr/(m*n);
201 end

```

```
202
203
204     end
205
206 end
```

visualization.m

```
1 function visualization
2
3 global f1;
4 f1 = figure('OuterPosition',[0 0 700 600]);
5
6 files = dir('data/*stm.mat');
7
8 for file=1:numel(files)
9
10    clear myplain mysm;
11    clf;
12
13    filename = files(file).name;
14
15    % extract parameter values
16    params = sscanf(filename, 'd%d_i%d_v%f_i%f_.mat');
17    dur = params(1);
18    inten = params(2);
19    vis = params(3);
20    importance = params(4);
21    location = 'fri';
22
23    plot_paths(filename);
24    [n t dist dist_std traveltime] = completed_paths(filename);
25
26    data = ...
27        sprintf('%i,%i,%f,%f,%i,%f,%f,%f',dur,inten,vis,importance,n,t,dist,dist_std,trave
28    disp(data);
29
30 end
31
32
33
34
35 %-----%
36 function [n t_norm dist dist_std traveltime] = completed_paths(filename)
37 load(strcat('data/',filename));
38
39 % number of paths completed
```

```

40     n = length(mysm.paths);
41
42     % travel time
43     t = zeros(3,n);
44
45     for i=1:n
46         path=mysm.paths(i);
47         start = path.coordinates(1,:);
48         dest = path.coordinates(end,:);
49         dist = sqrt(sum((dest-start).^2));
50
51         traveltimes = path.time;
52
53         t(:,i) = [dist;traveltimes;dist/traveltimes];
54     end
55
56     t_norm = mean(t(3,:));
57     dist = mean(t(1,:));
58     dist_std = std(t(1,:));
59     traveltimes = mean(t(2,:));
60
61 end
62
63
64
65 %-----
66 function plot_paths(filename)
67     global f1;
68     load(strcat('data/',filename));
69
70
71     % plot ground structure
72     subplot(1,2,1);
73     pathMax = 100;
74     title('Evolving Trails');
75
76     A=myplain.realGround;
77     B=myplain.ground-myplain.initialGround;
78
79     % shift B above the maximum of A
80     B_shifted = B-min(B(:))+max(A(:))+1;
81
82     % create fitted colormap out of two colormaps
83     range_A = max(A(:))-min(A(:));
84     range_B = max(B(:))-min(B(:));
85
86
87     % to adjust caxis and colormap
88     range_b = range_B;
89
90     for i=1:10

```

```

91      if (range_b>=pathMax/10*(i-1))&&(range_b<=pathMax/10*i)
92          range_B = pathMax/10*i;
93      end
94  end
95
96 % adjusting colormap
97
98 cm = [gray(ceil(64*range_A/range_B));flipud(summer(64))];
99
100 % plotting
101 pcolor(B_shifted)
102 shading interp
103 hold on
104 contour(A)
105 axis equal tight off ij;
106 colormap(cm)
107 caxis([min(A(:)) max(A(:))+range_B])
108
109 freezeColors % http://www.mathworks.com/matlabcentral/fileexchange/7943
110
111
112 % plot uncompleted vector paths
113 for ped=mysm.pedestrians
114     plot(ped.way(:,1), ped.way(:,2), 'r');
115 end
116
117 % plot completed vector paths
118 for path=mysm.paths
119     plot(path.coordinates(:,1),path.coordinates(:,2), 'b');
120 end
121
122 % plot cities
123 if(strfind(filename,'fri'))
124     p = [34 26;30 27;35 31;30 12;31 23;28 21;13 12;34 20;21 33;26 ...
125         35;27 26;21 19;21 29;14 34];
126 else
127     p = [23 31;29 23;34 32;27 28;17 40;32 19];
128 end
129 plot(p(:,1),p(:,2), 'wo');
130
131 hold off;
132
133
134
135 % plot travel time
136 subplot(1,2,2);
137 hold on;
138 for path=mysm.paths
139     plot(path.timeOfArrival,path.time,'o');
140     text(path.timeOfArrival,path.time,sprintf('%i',path.type));

```

```
141     end  
142  
143     axis square;  
144     hold off;  
145  
146  
147     saveas(f1,strcat('images2/',filename,'.png'));  
148  
149 end
```