Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Simulation of Human Trail Systems in Parks

Markus Frei & Adrian Gaemperli

Zurich
May 2011

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.

Markus Frei

Adrian Gaemperli

Contents

1 Individual contributions

The whole project was done in a cooperative manner.

2 Introduction and Motivations

We decided to simulate a park with some paths, an obstacle and different pedestrian destinations. Human trail systems can be described by very simple formulas. This was of special interest to us and it is interesting to simulate this very common activity of walking. Furthermore it was a perfect opportunity to improve our MATLAB programming skills.

Of high interest are the following questions:

- With which path schema do people walk at least on the grassland?
- In which location do people walk at least beside the path?
- With which setup do they walk at least beside the paths?
- With which combination they walk at least beside the paths?

3 Description of the Model

Our simulation model is based on the continuous model of the paper *Modeling the evolution of human trail systems* [HKM97]. We used the discretized model of the previous Project Report [PP10]. A short summary for the discretized model and explanation of our extended model will be given.

3.1 Summary of the model

The park is divided into a mesh of small squares. The different values are constant for each square. We use a model with a discrete time step. The ground structure is updated by the following formula taken from [PP10].

$$G(\mathbf{r}, t+1) = G(\mathbf{r}, t) + \frac{1}{T(\mathbf{r})}\left[G_0(\mathbf{r}) - G(\mathbf{r}, t)\right] + I(\mathbf{r})\left[1 - \frac{G(\mathbf{r}, t)}{G_{max}(\mathbf{r})}\right]\sum_{\alpha}\delta(\mathbf{r} - \mathbf{r}_{\alpha}(t)) \quad (1)$$

$G(\mathbf{r}, t)$	ground structure at place $\mathbf{r}$ for time t
$G_{max}(\mathbf{r}, t)$	maximal ground structure at place $\mathbf{r}$
$T(\mathbf{r})$	durability of a footprint at place $\mathbf{r}$
$I(\mathbf{r})$	intensity of a footprint at place $\mathbf{r}$

To calculate the attractiveness V_{tr} of a place r_α we use the formula taken from [PP10].

$$V_{tr}(\mathbf{r}_\alpha, t) = \frac{\sum\limits_{r \in \Omega} e^{\frac{-|\mathbf{r}-\mathbf{r}_\alpha|}{\sigma(\mathbf{r}_\alpha)}} G(\mathbf{r}, t)}{|\Omega|} \tag{2}$$

Ω	set of places which influence the attractiveness
$\sigma(\mathbf{r})$	visibility of a place $\mathbf{r}$

We use the following formula taken from [PP10] to calculate the direction which the pedestrian walks in this time step.

$$e_\alpha(\mathbf{r}_\alpha, t) = \rho * \frac{\mathbf{d}_\alpha - \mathbf{r}_\alpha}{\|\mathbf{d}_\alpha - \mathbf{r}_\alpha\|} + \frac{\arg\max_{\mathbf{r} \in \Lambda} V_{tr}(\mathbf{r}, t)}{\|\arg\max_{\mathbf{r} \in \Lambda} V_{tr}(\mathbf{r}, t)\|} \tag{3}$$

ρ If smaller than one the attractiveness gets more important. If larger than one the distance gets more important

3.2 Extensions

We had to extend this model because we faced several problems. The pedestrians walked through the obstacle. This limitation of the model was known from [PP10]. To avoid this behaviour we check every new position of a pedestrian if it is inside an obstacle. In this case we exclude this position from the possible new positions and guide the pedestrian around the obstacle. The position closest to the destination will be chosen. In some cases it is still possible that they are trapped. This will be shown during the discussion of the simulation. For each of the eight positions around the pedestrian we have to calculate the attractiveness. Two or more positions often have exactly the same value. To make a proper choice we took the position which is closest to the destination. The pedestrians are free to choose a position on a path which is in the closest direction to the destination. The attractiveness is not important if the pedestrians are on a path. For ρ equal to zero the pedestrians can now find a way to the destination and stay always on the paths. For ρ bigger than zero they leave at some places the paths and walk over the grassland as expected. But for some values of ρ it is still possible that the pedestrians are trapped on a place. The circumstances are explained in the discussion of the simulations.

4 Implementation

4.1 Visualisation of the results

The visualisations are a representation of the ground structure matrix. Blue circles represent entrances, yellow circles the position of a kiosk and green crosses pedestrians. Blue parts have a low ground structure value whereas red parts have a high ground structure value.

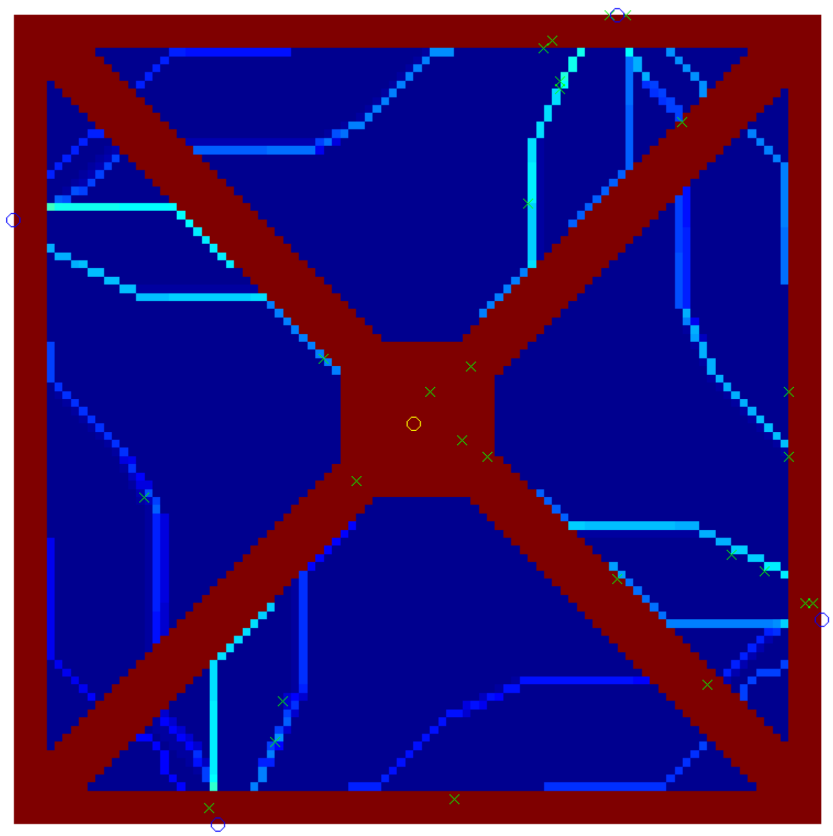


Figure 1: An example of a visualisation

4.1.1 Filename

The filename of the figures states the parameters which were used and the step number.

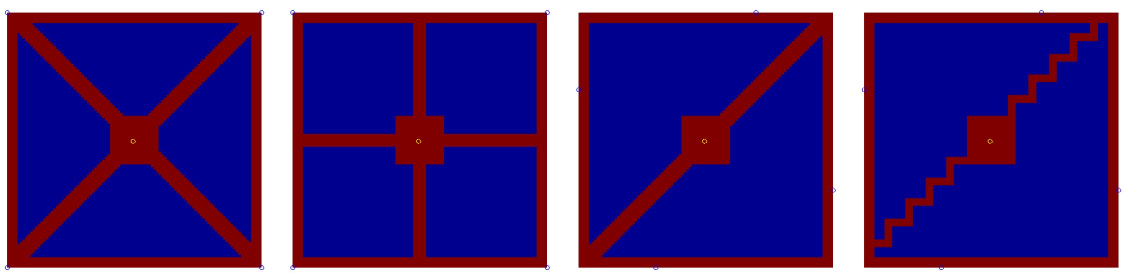
v1	program version
adjustment	position of the entrances, equals 0 if entrances are in the corners
setup	park setup number
loc	location number
path	path schema number
rho	value of ρ, which defines how the attractiveness and the shortest way are weighted
step	step number

Table 1: Explanation of the filename

4.2 Parameters

4.2.1 Path schemas

We developed several path schemas which seemed to be of special interest to us. All path schemas have a path around the park and around the obstacle.



(a) Schema 1 (b) Schema 2 (c) Schema 3 (d) Schema 4

Figure 2: Examples of path schemas

Schema 1 The Path schema 1 has two straight diagonal paths. (see Figure 2a)

Schema 2 At this schema the inner and outer paths are connected by paths in the middle of the sides. (see Figure 2b)

Schema 3 This schema is very similar to schema 1 but it has only one diagonal path. (see Figure 2c)

Schema 4 This schema consists of a jagged diagonal path. (see Figure 2d)

4.2.2 Park setup

Kiosk (1) The kiosk is in the middle of the park and it changes the pedestrians' behaviour. 20% of pedestrians walk to the kiosk and after that to their destination.

Lake (2) The lake is in the middle of the park and is a square.

4.2.3 Locations

As we realised that there is only a very little difference between location 1 and 2 we later focused on location 2.

Location 1 The pedestrians enter the park at one side and walk to one exit of the other side, uniformily distributed.

Location 2 The pedestrians walk from one entrance to another or walk back to the entrance, uniformily distributed.

4.3 Class design

We implemented the model making use of object orientation to make the code more readable. Furthermore we wrote a starter-script called *start.m* to make it easier to start the simulations with a single call of *start()*.

4.3.1 Simulation

The simulation class is responsible for the coordination of the simulation. It gives the "time-step" to the park and saves the figures, determines the start and endpoints of the pedestrians and adds them to the simulation. Furthermore it also determines if the pedestrians has to visit the kiosk.

4.3.2 Initialisation

This class is called by the simulation class and constructs the park. It adds the obstacles (the lake and the kiosk) and sets the paths of the path schema.

4.3.3 Park

The park class registers all pedestrians and passes on the time-step to the pedestrians. The class is responsible for the regeneration of the ground.

4.3.4 Pedestrian

In the pedestrian class all calculation is done which is needed to find out the walking direction and location of the pedestrians. There is one instance per pedestrian.

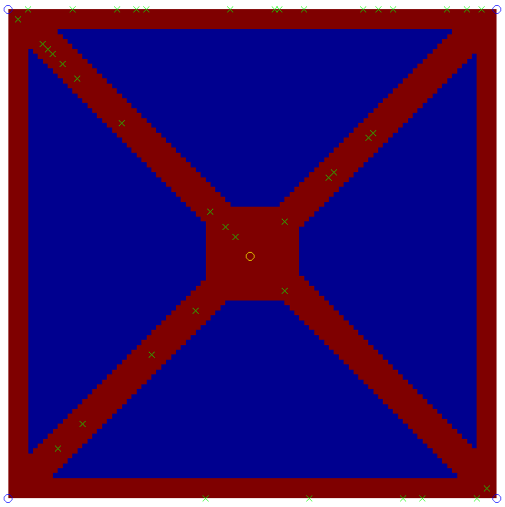
5 Simulation Results and Discussion

Firstly, we will show the results when the entrances are in the corners of the park. In this section we will describe the differences for the two path schemas, locations of the park and park setups. A special focus will be set on different values of ρ.

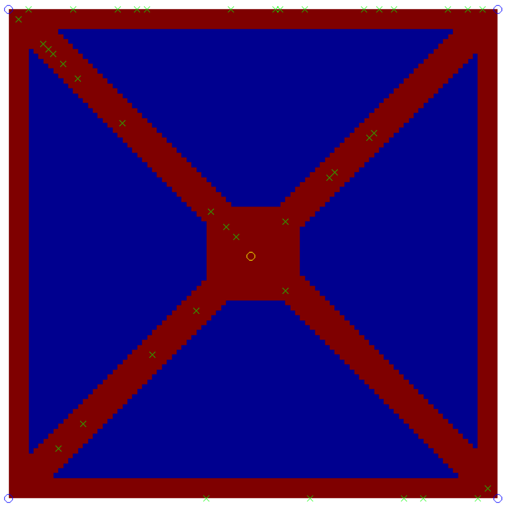
We will move the entrances to the centre of the park edges in two steps. In this part we focus on the second location and the setup with the kiosk. We will change the path schema and ρ. In the last section we experiment with alternative path schemas. For each case we simulated 500 steps. Our research has shown that there is no significant difference.

5.1 Entrances in the corners

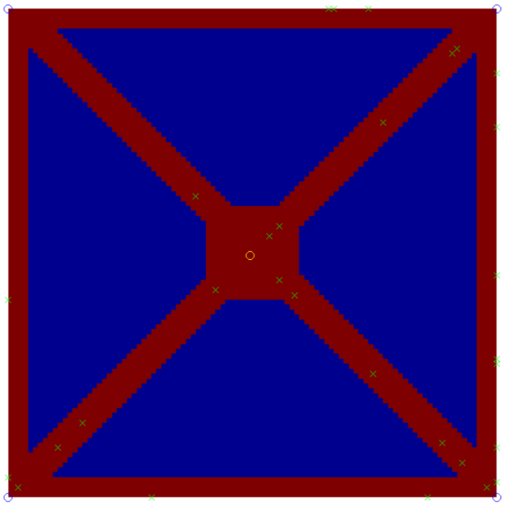
5.1.1 Path schema 1



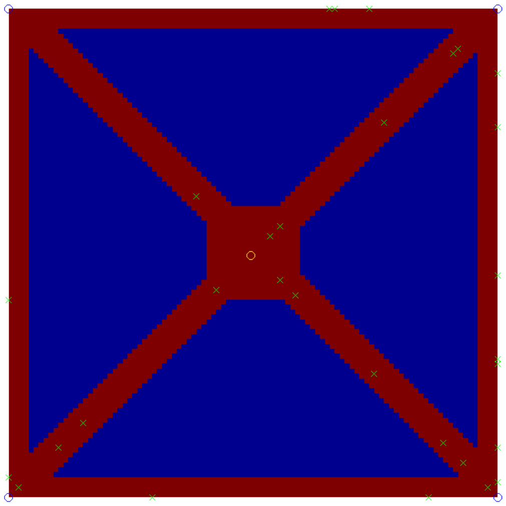
(a) adjustment: 0, park setup: 1, location: 1, path schema: 1, ρ: 1.2, step: 500



(b) adjustment: 0, park setup: 1, location: 1, path schema: 1, ρ: 1.6, step: 500



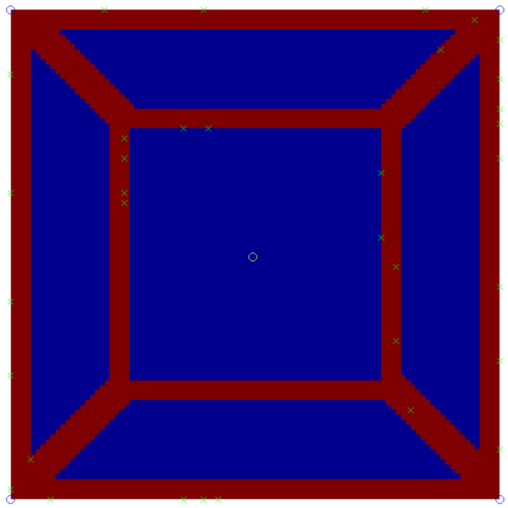
(c) adjustment: 0, park setup: 1, location: 2, path schema: 1, ρ: 0, step: 500



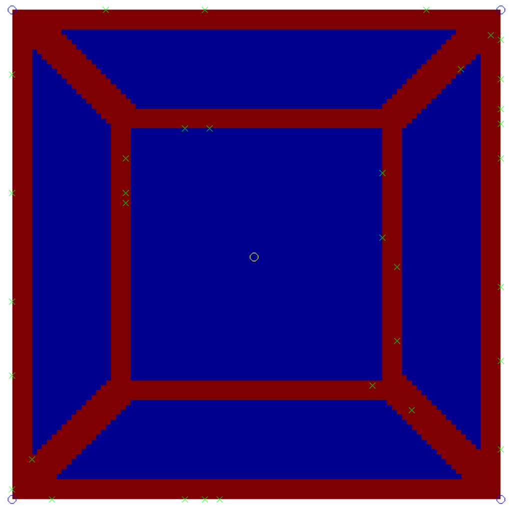
(d) adjustment: 0, park setup: 1, location: 2, path schema: 1, ρ: 5, step: 500

Figure 3

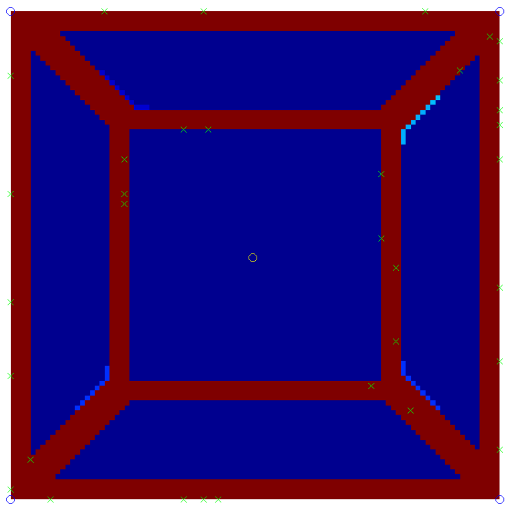
The pedestrians never leave the paths in the first path schema with the kiosk setup independent of the location of the park (see Figure 3). This is reasonable because they always have a path in the shortest direction to their destinations. This path schema in combination with these entrances would be optimal for park designers because the grassland would not be destroyed by people. If we change the park setup and increase ρ the pedestrians walk over the grassland (see Figure 4).



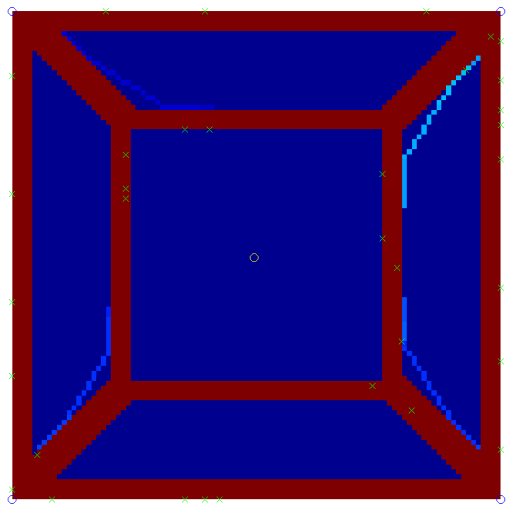
(a) adjustment: 0, park setup: 2, location: 2, path schema: 1, ρ: 0.8, step: 500



(b) adjustment: 0, park setup: 2, location: 2, path schema: 1, ρ: 1.2, step: 500



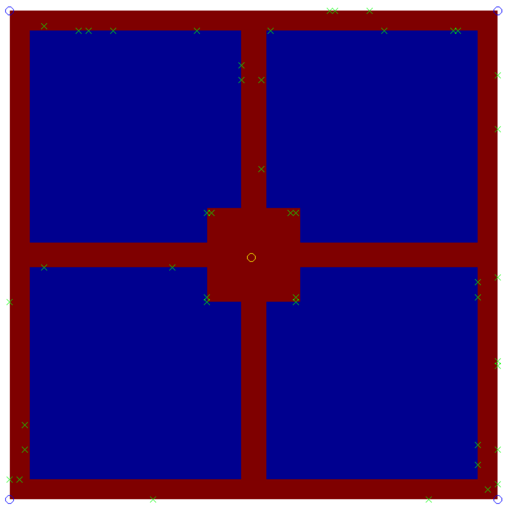
(c) adjustment: 0, park setup: 2, location: 2, path schema: 1, ρ: 1.6, step: 500



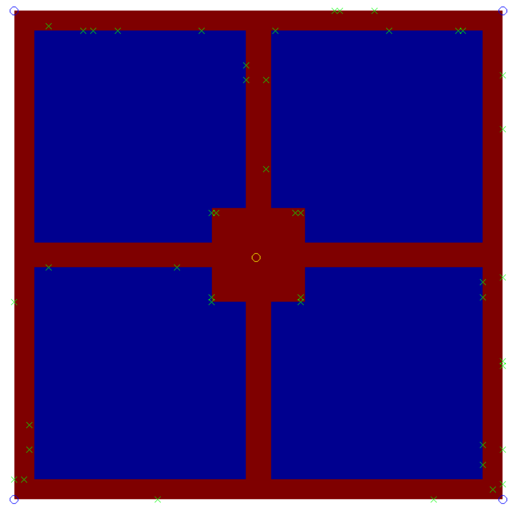
(d) adjustment: 0, park setup: 2, location: 2, path schema: 1, ρ: 5, step: 500

Figure 4

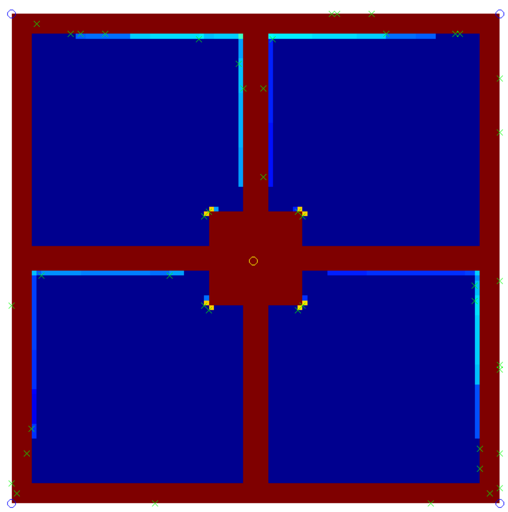
5.1.2 Path schema 2



(a) adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 0, step: 500



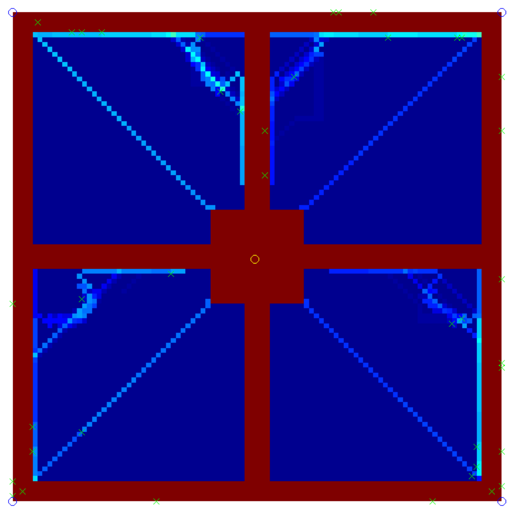
(b) adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 0.4, step: 500



(c) adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 0.8, step: 500

Figure 5

For little values of ρ we can see the trapped pedestrians in the figures (see Figure 5). The pedestrian can reach the kiosk. Now he walks in the direction of his chosen park exit. Until he reaches the corner of the squared region with high ground structure. Then he is trapped because of the little value of ρ. The direction of the park exit has a too low weight. It would be interesting to combine this algorithm with a path finding algorithm to avoid such problems and improve the model of human trail system.



(a) adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 1.2, step: 500

Figure 6

For ρ equal to 0.8 the pedestrians start to walk over the grassland. But the paths are too attractive that is why they are still bound to them. For increasing ρ the pedestrians walk more and more beside the paths (see Figure 6). The pedestrians who want to go to the kiosk leave the paths earlier than those who want to go to the diagonal entrance. The paths in the diagonals are made by the pedestrians who walked from the kiosk in the centre to the entrances in the corners of the park.

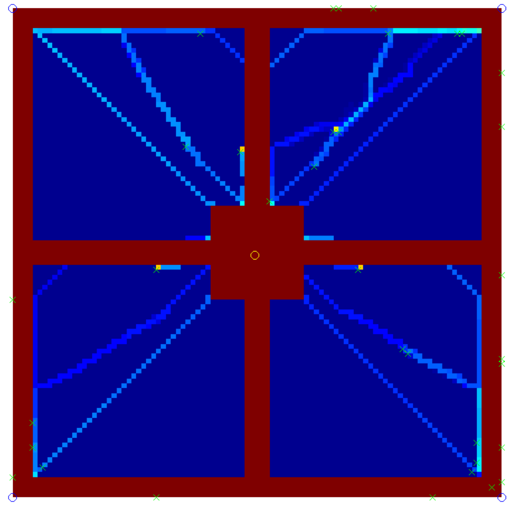


Figure 7: adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 1.6, step: 500

These effects continue for ρ equal to 1.6 (see Figure 7). In this picture we can see three regions where the pedestrians are trapped (yellow dots). We can not explain this effect because ρ is bigger than one and because of this the pedestrians should find their way to the entrance. Beside this we can see in the upper right part of the park that two paths are merged together.

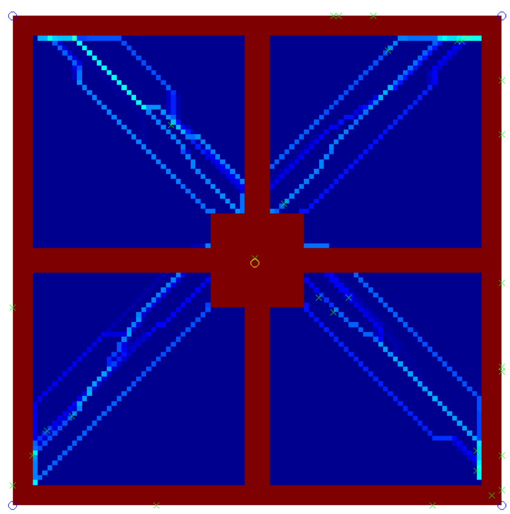


Figure 8: adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 2.0, step: 500

For ρ equal to 2 the paths are really close together (see Figure 8). This effect gets more obvious the more we increase ρ. For ρ equal to 2.4 the pedestrians only walk on the diagonals, the shortest way to their destinations (see Figure 9).

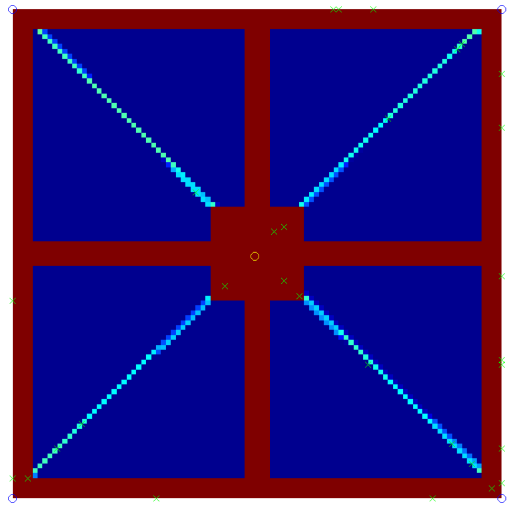


Figure 9: adjustment: 0, park setup: 1, location: 2, path schema: 2, ρ: 2.4, step: 500

Figure 9 shows the optimal path schema when the entrances are in the corners of the park.

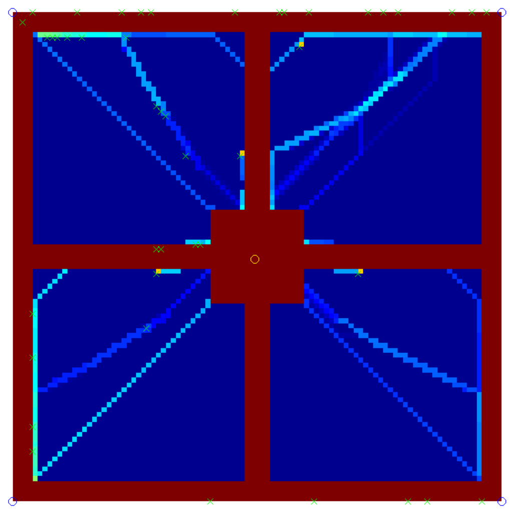
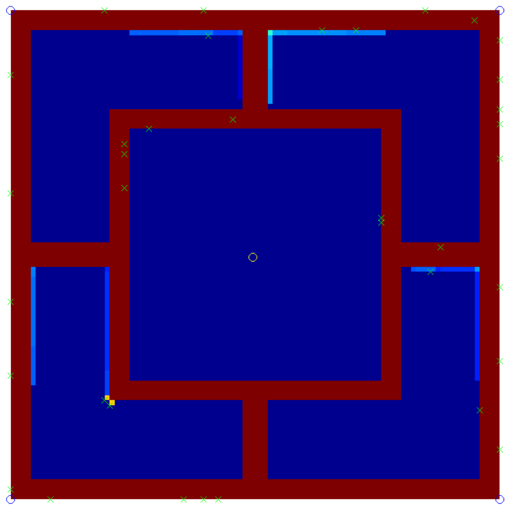
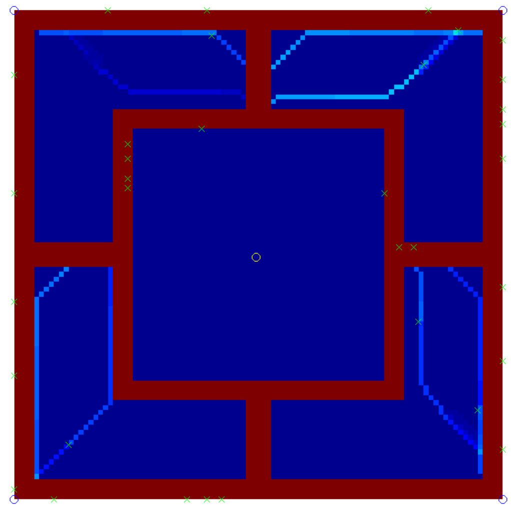


Figure 10: adjustment: 0, park setup: 1, location: 1, path schema: 2, ρ: 1.6, step: 500

If we set another location and keep the kiosk setup it does not change anything (see Figure 10). As a consequence we do not have to change ρ in this case.



(a) adjustment: 0, park setup: 2, location: 2, path schema: 2, ρ: 0.8, step: 500



(b) adjustment: 0, park setup: 2, location: 2, path schema: 2, ρ: 1.6, step: 500

Figure 11

The lake setup is not as interesting as the kiosk setup (see Figure 11). This is why we do not observe this case in the future analysis. We will focus on the kiosk setup and the second location.

5.2 Entrances shifted 25% of the edges length

The entrances of the park are now shifted towards the centre of the park edges.

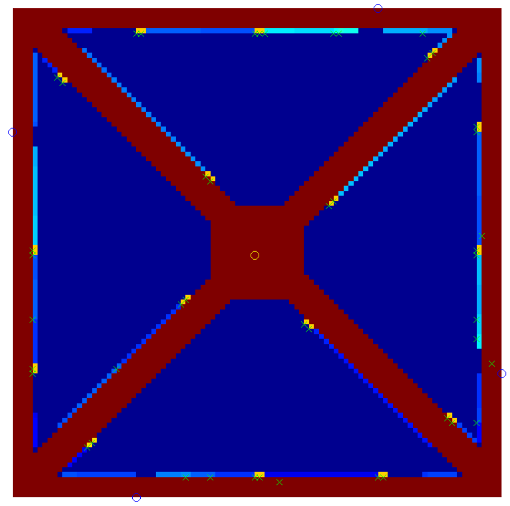
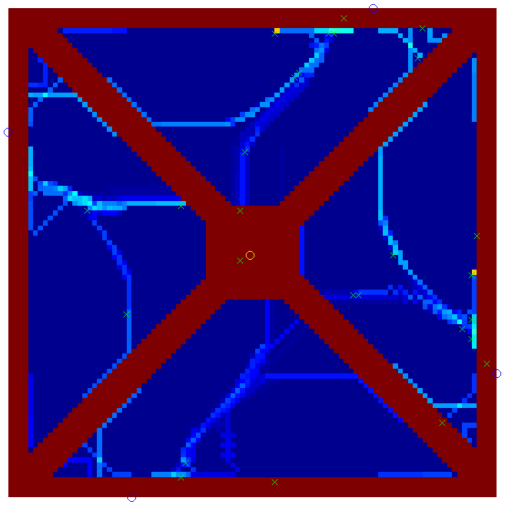
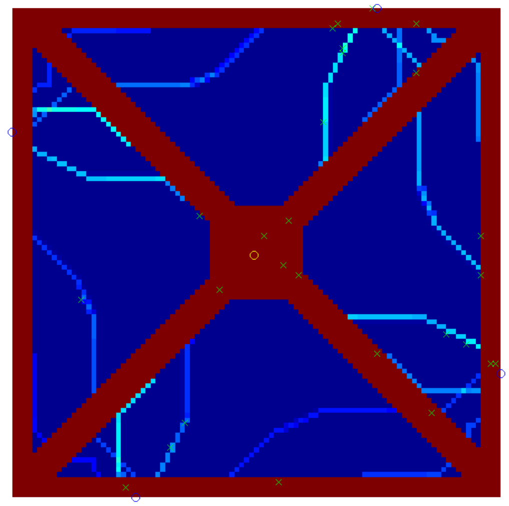


Figure 12: adjustment: 25, park setup: 1, location: 2, path schema: 1, ρ: 0.8, step: 500

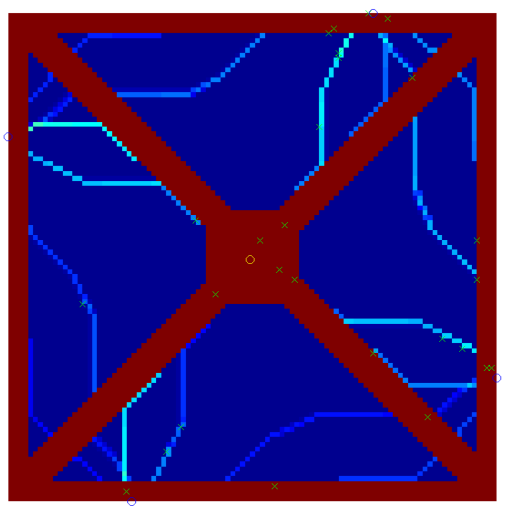
For ρ equal to 0.8 there are a lot of places where the pedestrians are trapped (see Figure 12). This is the case because the direction has a low weight. We can not explain why there are such places on the diagonal paths. If we increase ρ more and more grassland is destroyed by the pedestrians (see Figure 13).



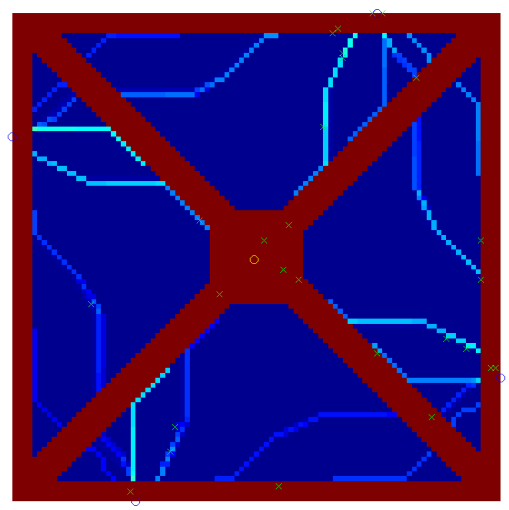
(a) adjustment: 25, park setup: 1, location: 2, path schema: 1, ρ: 1.2, step: 500



(b) adjustment: 25, park setup: 1, location: 2, path schema: 1, ρ: 1.6, step: 500



(c) adjustment: 25, park setup: 1, location: 2, path schema: 1, ρ: 2.0, step: 500



(d) adjustment: 25, park setup: 1, location: 2, path schema: 1, ρ: 2.4, step: 500

Figure 13

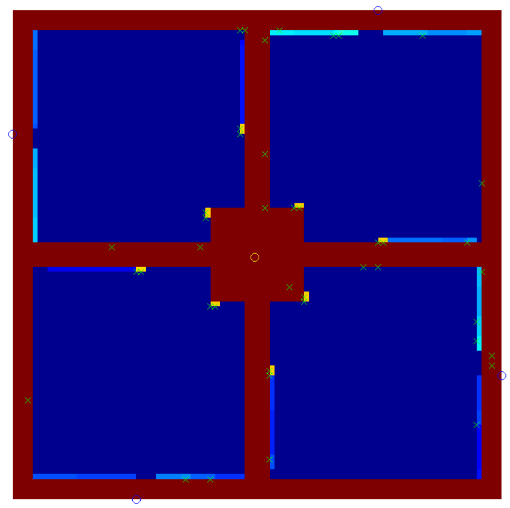
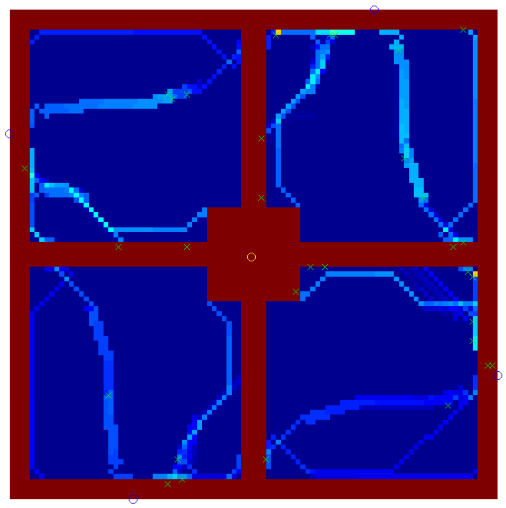
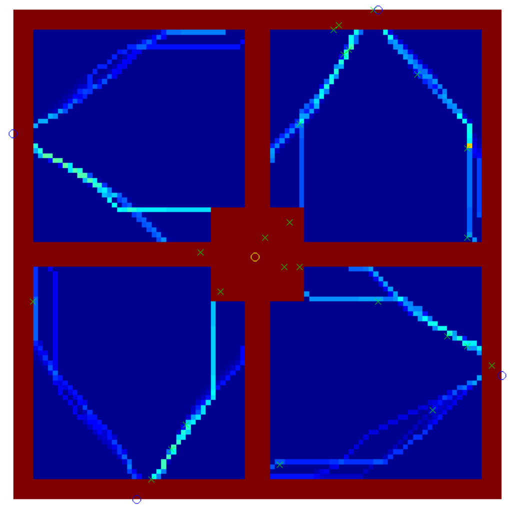


Figure 14: adjustment: 25, park setup: 1, location: 2, path schema: 2, ρ: 0.8, step: 500

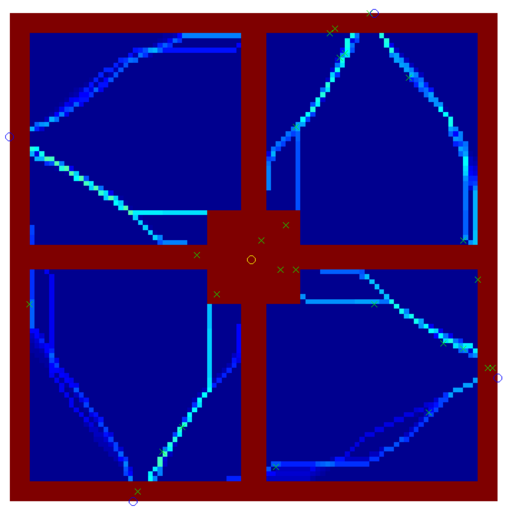
For the second path schema and little values of ρ there are also places where the pedestrians are trapped (see Figure 14). If we increase ρ the pedestrians walk a lot over the grassland again (see Figure 15).



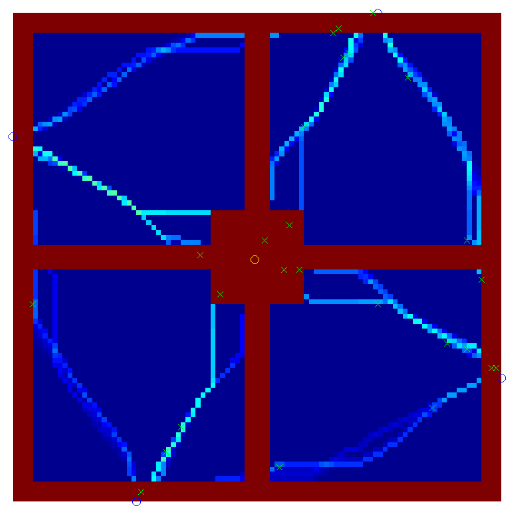
(a) adjustment: 25, park setup: 1, location: 2, path schema: 2, ρ: 1.2, step: 500



(b) adjustment: 25, park setup: 1, location: 2, path schema: 2, ρ: 1.6, step: 500



(c) adjustment: 25, park setup: 1, location: 2, path schema: 2, ρ: 2.0, step: 500



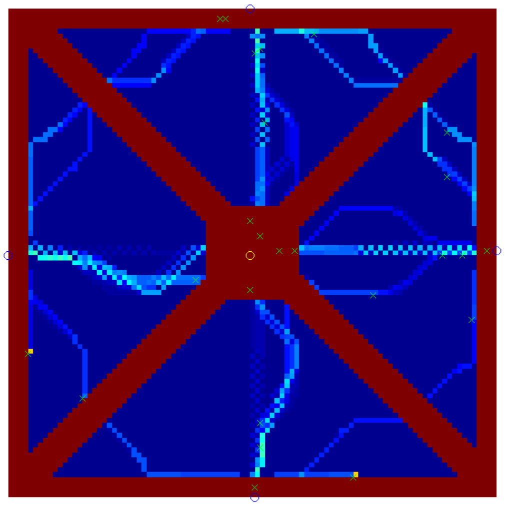
(d) adjustment: 25, park setup: 1, location: 2, path schema: 2, ρ: 2.4, step: 500

Figure 15

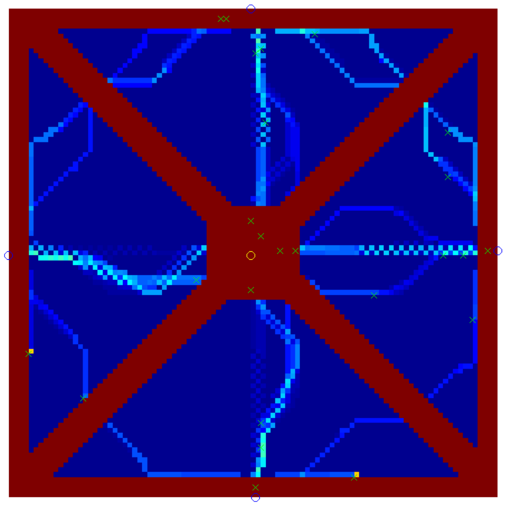
For both path schemas there are a lot of small trails. Each trail can be dedicated to a destination and a starting point of the pedestrian.

5.3 Entrances in the middle of the park edges

The entrances of the park are now shifted to the centre of the park edges.



(a) adjustment: 50, park setup: 1, location: 2, path schema: 1, ρ: 0.8, step: 500



(b) adjustment: 50, park setup: 1, location: 2, path schema: 1, ρ: 1.2, step: 500

Figure 16

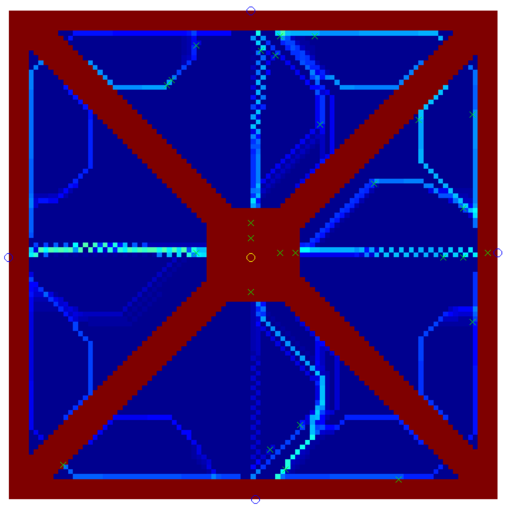
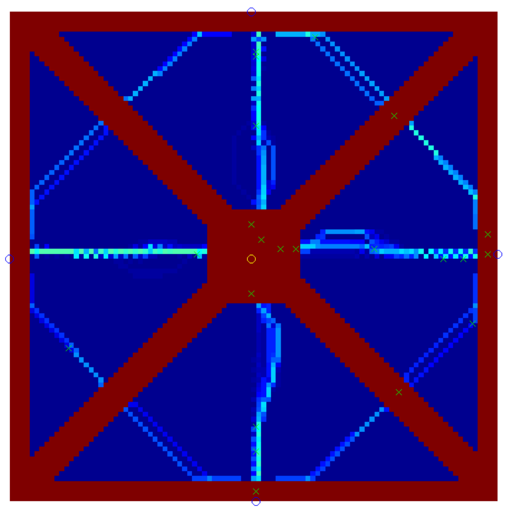
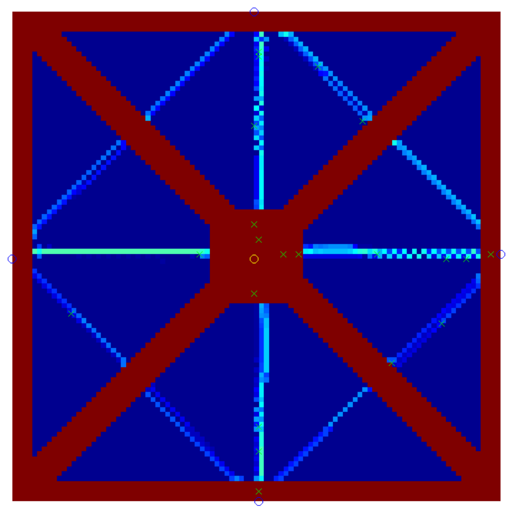


Figure 17: adjustment: 50, park setup: 1, location: 2, path schema: 1, ρ: 1.6, step: 500

The results for ρ equals to 1.2 are not how we expected them (see Figure 16). If we set a high value for ρ we get what we expected (see Figure 18).

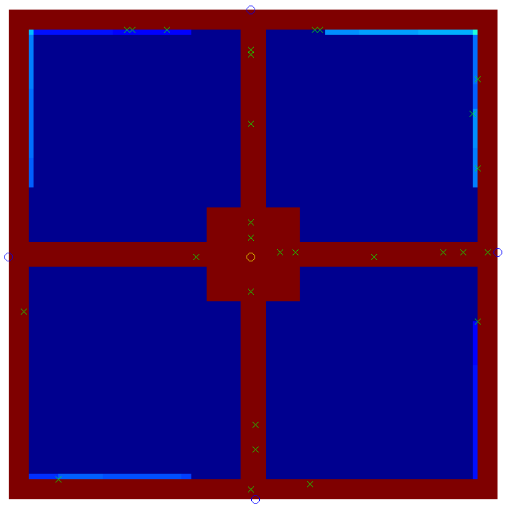


(a) adjustment: 50, park setup: 1, location: 2, path schema: 1, ρ: 2.0, step: 500

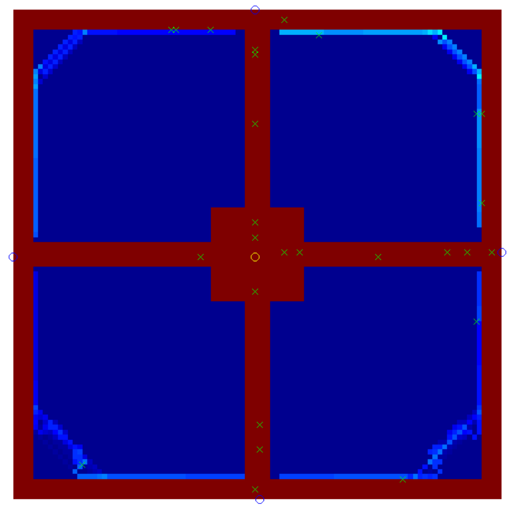


(b) adjustment: 50, park setup: 1, location: 2, path schema: 1, ρ: 2.4, step: 500

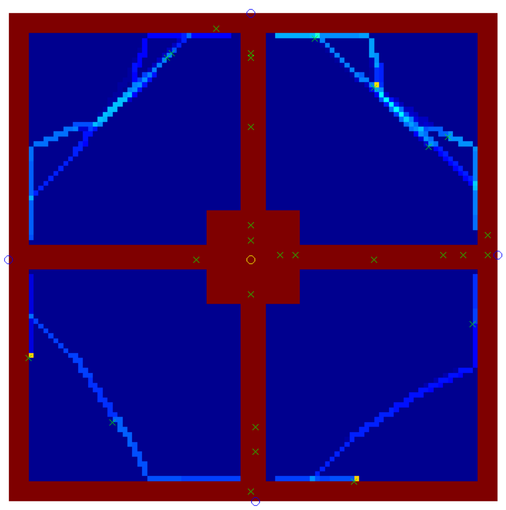
Figure 18



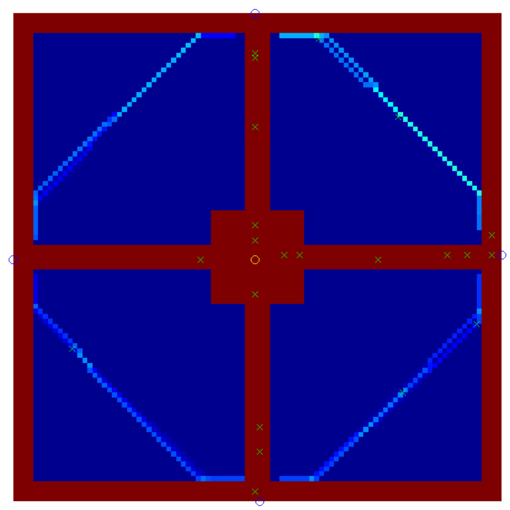
(a) adjustment: 50, park setup: 1, location: 2, path schema: 2, ρ: 0.8, step: 500



(b) adjustment: 50, park setup: 1, location: 2, path schema: 2, ρ: 1.2, step: 500



(c) adjustment: 50, park setup: 1, location: 2, path schema: 2, ρ: 1.6, step: 500



(d) adjustment: 50, park setup: 1, location: 2, path schema: 2, ρ: 2.0, step: 500

Figure 19

In this set of figures (see Figure 19) we can see how the pedestrians walk more and more over the grassland for increasing ρ until they walk the shortest way to their destination. This leads to the optimal path schema for this case (see Figure 20).

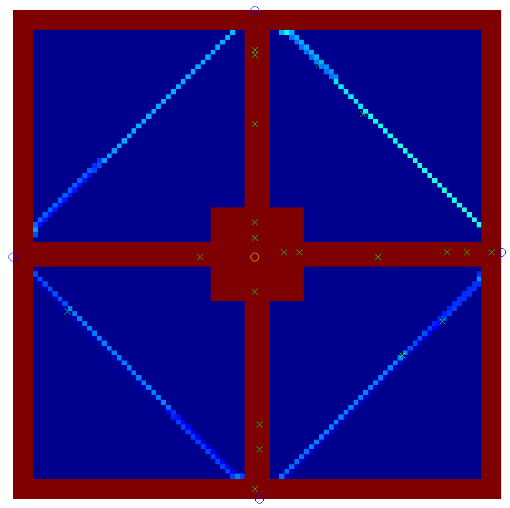
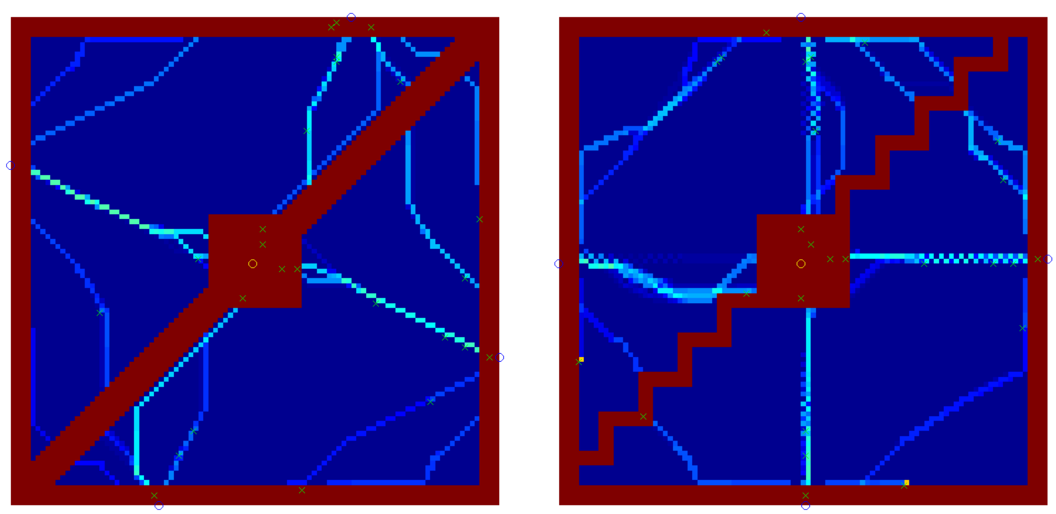


Figure 20: adjustment: 50, park setup: 1, location: 2, path schema: 2, ρ: 2.4, step: 500

5.4 Alternative path schemas



(a) adjustment: 30, park setup: 1, location: 2, path schema: 3, ρ: 1.6, step: 500

(b) adjustment: 50, park setup: 1, location: 2, path schema: 4, ρ: 1.6, step: 500

Figure 21

We experimented with alternative path schemas. But they do not differ much from the already discussed results, but these figures are illustrative.

6 Summary and Outlook

It can be easily seen that the pedestrians never leave the paths with path schema 1, a kiosk and zero adjustment. The extended model works fine but there are still problems in some cases. However, a combination of this model with a path finding algorithm would solve most of them. With such a combination it is possible to simulate more complex environments. Nevertheless, we got reasonable results and we could find optimal path schemas for different entrances.

An interesting enhancement of the project would be an easy visualisation of which routes have changed which ground. Moreover to visualize the usage of the paths. Furthermore our simulation was related to the different values of ρ. But it would also be very interesting to change the durability and intensity. Another possible interesting extension would be that the paths' ground structure does not change abruptly.

7 References

[HKM97] Dirk Helbing, Joachim Keltsch & Peter Molnar: *Modeling the evolution of human trail systems*, Nature 388 (1997), 47-50

[PP10] Jonas Pfefferle & Nicholas Pleschko: *Simulation of Human Trail Systems*, Modelling and Simulating Social Systems with MATLAB, ETH Project Report HS2010, December 13, 2010

A MATLAB code

A.1 start.m

```
function start()
    %starts all simulation which are defined in simMatrix and saves the
    %results in the folder ./output
```

```
close all hidden;
clear;

%first parameter: ParkSetup; second parameter: Location; third
%parameter: PathSchema 4. Adjustment 5. rho

simMatrix = [
    1,1,1,0,1.2;
    1,1,1,0,1.6;
    1,1,2,0,1.6;
    1,2,1,0,0;
    1,2,1,0,5;
    1,2,2,0,0.4;
    1,2,2,0,0.8;
    1,2,2,0,0;
    1,2,2,0,1.2;
    1,2,2,0,1.6;
    1,2,2,0,2.4;
    1,2,2,0,2.0;
    2,2,1,0,0.8;
    2,2,2,0,0.8;
    2,2,1,0,1.2;
    2,2,2,0,1.2;
    2,2,1,0,1.6;
    2,2,2,0,1.6;
    2,2,1,0,5;
    1,2,1,25,0.8;
    1,2,1,25,1.2;
    1,2,1,25,1.6;
    1,2,1,25,2;
    1,2,1,25,2.4;
    1,2,2,25,0.8;
    1,2,2,25,1.2;
    1,2,2,25,1.6;
    1,2,2,25,2;
    1,2,2,25,2.4;
    1,2,1,50,0.8;
    1,2,1,50,1.2;
    1,2,1,50,1.6;
    1,2,1,50,2;
    1,2,1,50,2.4;
    1,2,2,50,0.8;
    1,2,2,50,1.2;
    1,2,2,50,1.6;
    1,2,2,50,2;
    1,2,2,50,2.4;
    1,2,3,30,1.2;
    1,2,3,30,1.6;
    1,2,3,30,2.0;
```

```
        1,2,3,50,1.2;
        1,2,3,50,1.6;
        1,2,3,50,2.0;
        1,2,4,30,1.2;
        1,2,4,30,1.6;
        1,2,4,30,2.0;
        1,2,4,50,1.2;
        1,2,4,50,1.6;
        1,2,4,50,2.0
        ];

    for i = 1:size(simMatrix,1)
        simulation = Simulation(simMatrix(i,1), simMatrix(i,2), ...
            simMatrix(i,3) , simMatrix(i,4), simMatrix(i,5));
        simulation.start()
    end

    clear;
end
```

A.2 Simulation.m

```
classdef Simulation < handle
    %SIMULATION controls the simulation
    %

    properties
        park;
        printInterval = 50;
        numberOfSteps = 500;
        pedRate = 0.4; %in this number of steps is one pedestrian ...
            entering the park, set 0 for only one pedestrian (testing)
        location;
        parkSetup;
        pathSchema
        step = 0;
        seedValue;
        adjustment;
        rho;
    end

    properties(Constant = true)
        DAY_LENGTH = 100;

    end

    methods
```

```
        function this = Simulation(simParkSetup, simLocation, ...
            simPathSchema, simAdjustment, simRho)

            %Set random seed
            this.seedValue = 4242;
            RandStream.setDefaultStream(RandStream('mt19937ar', 'seed', ...
                this.seedValue));

            this.parkSetup = simParkSetup;
            this.location = simLocation;
            this.pathSchema = simPathSchema;
            this.adjustment = simAdjustment;
            this.rho = simRho;

            init = Initialisation(simParkSetup, simPathSchema, ...
                this.adjustment);
            this.park = init.getPark();
        end

        function start(this)
            this.saveData();

            disp('Simulation started');

            for step = 1:this.numberOfSteps
                this.step = step;

                %Information
                %{
                disp(strcat('SIMULATION_STEP_', num2str(this.step)));
                disp('number of pedestrians =');
                disp(length(this.park.pedestrians));
                %}

                %generate pedestrians and put them in the park
                if isequal(this.pedRate, 0)
                    this.generatePedestrian();
                    this.pedRate = 2;

                elseif rand(1) <= this.pedRate && this.pedRate <= 1
                    this.generatePedestrian(this.adjustment);

                end

                %make a simulation step for de park
                %pedestrians will be updated in this function
                this.park.step();

                if mod(step, this.printInterval) == 0
                    % print groundstructure and attractivness
```

```
                    this.saveData();
                end

            end


            disp('Simulation completed');
            return
        end

        function generatePedestrian(this, adjus)
            %generates the pedestrians and adds them to the park

            %set entrances and exits of the park
            entrance1 = [1,1+adjus];
            entrance2 = [length(this.park.groundStructure)-adjus,1];
            entrance3 = [1+adjus, length(this.park.groundStructure)];
            entrance4 = [length(this.park.groundStructure), ...
                length(this.park.groundStructure)-adjus];
            kioskPosition = [length(this.park.groundStructure)/2, ...
                length(this.park.groundStructure)/2];%when change ...
                remember to change also in the park

            %generate pedestrian for the different simulation types
            ped = Pedestrian();
            ped.setRho(this.rho);
            ped.setPark(this.park);

            if this.location == 1
                %At this location, in the morning all people want to ...
                    walk from
                %South to North (from the 2 entrances in the South (each ...
                    50%)
                %to the entrances in the North (each 50%) because ...
                    there's a
                %living district in the south and and work space in the ...
                    north.
                %In the evening vice versa.

                %in the morning the pedestrians walk from one side to the
                %other
                if this.isMorning()

                    %set start place of the pedestrian
                    if rand(1) <= 0.5
                        ped.setStart(entrance1);
                    else
                        ped.setStart(entrance2);
                    end

```

```
                    %there is a lake in the center
                    if this.parkSetup == 2

                    %there is a kiosk in the center
                    elseif this.parkSetup == 1
                        %set the kiosk as destination
                        if rand(1) <= 0.2
                            ped.addDestination(kioskPosition);
                        end
                    else
                        assert(false)
                    end

                    %set destination of the pedestrian
                    if rand(1) <= 0.5
                        ped.addDestination(entrance3);
                    else
                        ped.addDestination(entrance4);
                    end

                %in the evening the pedestrains walk in the other direction
                else
                    %set start place of the pedestrian
                    if rand(1) <= 0.5
                        ped.setStart(entrance3);
                    else
                        ped.setStart(entrance4);
                    end

                    if this.parkSetup == 2

                    elseif this.parkSetup == 1
                        %set the kiosk as destination
                        if rand(1) <= 0.2
                            ped.addDestination(kioskPosition);
                        end
                    else
                        assert(false)
                    end

                    %set destination of the pedestrian
                    if rand(1) <= 0.5
                        ped.addDestination(entrance1);
                    else
                        ped.addDestination(entrance2);
                    end

                end

            elseif this.location == 2
```

```
                %At this location, people are coming from every entrance
                %(each 25%) and they leave the park on every entrance with
                %probability 25%, because it's in the middle of the city.

                prob = rand(1);

                %set start place of the pedestrian
                if prob <= 0.25
                    ped.setStart(entrance1);
                    %disp('START AT [1,1]');
                elseif prob > 0.25 && prob <= 0.5
                    ped.setStart(entrance2);
                    %disp('START AT [length(this.park.groundStructure), ...
                        1]');
                elseif prob > 0.5 && prob <= 0.75
                    ped.setStart(entrance3);
                    %disp('START AT [1,length(this.park.groundStructure)]');
                else
                    ped.setStart(entrance4);
                    %disp('START AT [length(this.park.groundStructure), ...
                        length(this.park.groundStructure)]');
                end

                %there is a lake in the center
                if this.parkSetup == 2

                %there is a kiosk in the center
                elseif this.parkSetup == 1
                    %set the kiosk as destination

                    if rand(1) <= 0.2
                        ped.addDestination(kioskPosition);
                    end
                else
                    assert(false)
                end

                prob = rand(1);

                %set destination of the pedestrian
                if prob <= 0.25
                    ped.addDestination(entrance1);

                elseif prob > 0.25 && prob <= 0.5
                    ped.addDestination(entrance2);

                elseif prob > 0.5 && prob <= 0.75
                    ped.addDestination(entrance3);

                else
```

```
                    ped.addDestination(entrance4);

                end

            else
                assert(false)
            end

            %add pedestrian to the park
            this.park.pedestrians = [this.park.pedestrians ped];

        end

        function val = isMorning(this)
            %return: true or false
            %calculation: we set true if test is even, otherwise false

            test = floor(this.step/this.DAY_LENGTH);

            if ( rem(test,2) == 0)
                val = true;
            else
                val = false;
            end

        end

        function saveData(this)
            fig = figure(1);
            clf('reset');

            this.park.printMaps();

            filenamePart = strcat('v1_', 'adjustment', ...
                int2str(this.adjustment) ,'_setup', ...
                int2str(this.parkSetup), '_loc', int2str(this.location), ...
                '_path', int2str(this.pathSchema), '_rho', ...
                num2str(this.rho), '_step', num2str(this.step));

            saveas(fig, ['output/', filenamePart, '.png'])
            clf('reset')
            %close 1;

        end

    end

end
```

A.3 Initialisation.m

```
classdef Initialisation < handle
    %INITIALISATION initialises the simulation

    properties (GetAccess = 'private', SetAccess = 'private')
        pathSchema;
        parkSetup;
        park;
    end

    properties (Constant = true)
        PARK_SETUP_OBSTACLE_SIDELENGTH = [10, 50] % value has to be even
        PATH_WIDTH = 4
        PATH_GROUND_STRUCTURE = 150
        PARK_SIDE_LENGTH = 100
        KIOSK_GROUND_STRUCTURE = 150
    end

    methods
        function this = Initialisation(initParkSetup, initPathSchema, adjus)
            this.parkSetup = initParkSetup;
            this.pathSchema = initPathSchema;
            this.park = Park(this.PARK_SIDE_LENGTH, adjus);
        end

        function park = getPark(this)

            this.buildPathSchema();
            this.buildParkSetup();
            park = this.park;
            return
        end
    end

    methods(Access = private)

        function buildParkSetup(this)
            mid = floor(this.PARK_SIDE_LENGTH/2);
            obstacleHalfLength = ...
                floor(this.PARK_SETUP_OBSTACLE_SIDELENGTH(this.parkSetup)/2);
            switch this.parkSetup
                case 1
                    %A square kiosk in the middle of the park.
                    this.constructRectangle((mid-obstacleHalfLength):(mid+obstacleHalfLength),
                        this.KIOSK_GROUND_STRUCTURE);
                case 2
                    %A square lake in the middle of the park.
```

```matlab
                    this.constructRectangle((mid-obstacleHalfLength):(mid+obstacleHalfLength),
                        -1);
                otherwise
                    assert(false);
            end


        end


        function buildPathSchema(this)

            % draw path around park
            this.constructRectangle(1:this.PARK_SIDE_LENGTH, ...
                1:this.PATH_WIDTH, this.PATH_GROUND_STRUCTURE);
            this.constructRectangle(1:this.PARK_SIDE_LENGTH, ...
                this.PARK_SIDE_LENGTH-this.PATH_WIDTH:this.PARK_SIDE_LENGTH, ...
                this.PATH_GROUND_STRUCTURE);
            this.constructRectangle(1:this.PATH_WIDTH, ...
                1:this.PARK_SIDE_LENGTH, this.PATH_GROUND_STRUCTURE);
            this.constructRectangle(this.PARK_SIDE_LENGTH-this.PATH_WIDTH:this.PARK_SIDE_LENGT
                1:this.PARK_SIDE_LENGTH, this.PATH_GROUND_STRUCTURE);

            % draw path around obstacle
            p = ...
                floor(this.PATH_WIDTH+this.PARK_SETUP_OBSTACLE_SIDELENGTH(this.parkSetup)/2);
            mid = floor(this.PARK_SIDE_LENGTH/2);
            this.constructRectangle((mid-p):(mid+p), (mid-p):(mid+p), ...
                this.PATH_GROUND_STRUCTURE);


            switch this.pathSchema
                case 1
                    %path around the obstacle in the middle. A path around
                    %the park and a path from the corners of the park to
                    %the corresponding corner of the obstacle.

                    for i = 1:(this.PARK_SIDE_LENGTH-this.PATH_WIDTH)
                        this.constructRectangle(i:i+this.PATH_WIDTH,i:i+this.PATH_WIDTH,this.F
                        this.constructRectangle(this.PARK_SIDE_LENGTH+1-i-this.PATH_WIDTH:this
                    end


                case 2
                    %path around the obstacle in the middle. A path around
                    %the park and in the middle of the sides we dig a path
                    %to the middle of the corresponding middle of the side
                    %of the obstacle
                    halfPathWidth = floor(this.PATH_WIDTH/2);
```

```
                    this.constructRectangle((mid-halfPathWidth):(mid+halfPathWidth),1:this.PA
                        this.PATH_GROUND_STRUCTURE)
                    this.constructRectangle(1:this.PARK_SIDE_LENGTH,(mid-halfPathWidth):(mid+h
                        this.PATH_GROUND_STRUCTURE)
                case 3
                    % only one diagonal
                    for i = 1:(this.PARK_SIDE_LENGTH-this.PATH_WIDTH)
                        this.constructRectangle(i:i+this.PATH_WIDTH,i:i+this.PATH_WIDTH,this.P
                    end
                case 4
                    u = 0;

                    l = ceil(2*sqrt(this.PATH_WIDTH/4));
                    for i = ...
                        (2*this.PATH_WIDTH)+1:(this.PARK_SIDE_LENGTH-this.PATH_WIDTH)
                        if u == 0

                            this.constructRectangle(i:i+2*this.PATH_WIDTH,i:i+l,this.PATH_GROU
                            this.constructRectangle(i:i+l,i-2*this.PATH_WIDTH:i,this.PATH_GROU
                        end



                        u=u+1;
                        u = mod(u, 2*this.PATH_WIDTH);
                    end



                otherwise
                    assert(false);
             end

        end

        function constructRectangle(this, rows, columns, value)
            this.park.groundStructure(rows, columns) = value;
            this.park.groundStructureInit(rows, columns) = value;
            this.park.groundStructureMax(rows, columns) = value;
        end

    end

end
```

A.4 Park.m

```
classdef Park < handle
    %Park organises all informations about the Park.
    %   the function updateGroundStructure() updates the groundStructure

    properties %(SetAccess = private)
        groundStructure; % G(r, now)
        groundStructureMax; % G_max(r)
        groundStructureInit; % G_0(r) = G(r,0)
        intensity; % I(r)
        durability; % T(r)
        visibility; %   (r)
        pedestrians = [];%vector which contains all pedestrians
        currentStep = 0;
        adjustment;%only important for drawing the entrances on the map
    end

    properties(Constant = true)
        STANDARD_MAX_GROUNDSTRUCTURE = 100;
        STANDARD_VISIBILITY = 2;
        STANDARD_INTENSITY = 10;
        STANDARD_DURABILITY = 200;
    end

    methods

        function this = Park(dimension, adjus)
            this.groundStructure = ones(dimension);
            this.groundStructureInit = ones(dimension);
            this.groundStructureMax = ones(dimension) * ...
                this.STANDARD_MAX_GROUNDSTRUCTURE;
            this.intensity = ones(dimension) * this.STANDARD_INTENSITY;
            this.durability = ones(dimension) * this.STANDARD_DURABILITY;
            this.visibility = ones(dimension) * this.STANDARD_VISIBILITY;
            this.adjustment = adjus;
        end

        function setPedestrianGenerationRate(this,rate)
            this.pedestrianGenerationRate = rate;
        end

        function addPedestrian(this, object)
            object.setPark(this);
            object.setRho(this.STANDARD_RHO);
            this.pedestrians = [this.pedestrians object];
        end


        function updateGroundStructure(this)
            %updates the ground structure. The formula is separated into 2
            %parts.
```

```
        %change ground structure of every position in the park ...
            (regeneration)
        for i = 1:max(size(this.groundStructure))
            for j = 1:max(size(this.groundStructure))

                this.groundStructure(i,j) = ...
                    this.groundStructure(i,j) + ...
                    1/this.durability(i,j) * ...
                    (this.groundStructureInit(i,j)-this.groundStructure(i,j));
            end
        end

        %iterate over all pedestrians and add their footprints ...
            (destruction)
        for n = 1:length(this.pedestrians)
            pos = this.pedestrians(n).currentPosition;

            this.groundStructure(pos(1),pos(2)) = ...
                this.groundStructure(pos(1),pos(2)) + ...
                this.intensity(pos(1),pos(2)) * (1 - ...
                this.groundStructure(pos(1),pos(2))/this.groundStructureMax(pos(1),pos(2))

        end
    end

    function printGroundStructureMap(this)
        title(['ground structure (step '  int2str(this.currentStep)  ...
            ')'])
        axis([1 size(this.groundStructure,1) 1 ...
            size(this.groundStructure,2)])
        caxis([0, max(max(this.groundStructureMax))])


        %shading interp;
        pc = pcolor(this.groundStructure);
        set(pc,'edgecolor','none');



    end


    function printPedestriansMap(this)
        pedestriansPositions = zeros(length(this.pedestrians),2);
        for i=1:length(this.pedestrians)
            pedestriansPositions(i,:) = ...
                this.pedestrians(i).getPosition();
        end

```

```
            if ~isempty(pedestriansPositions)
                plot(pedestriansPositions(:,2),pedestriansPositions(:,1),'gx');
            end

            %plot the entrances of the park and kiosk position
            entrance1 = [1,1+this.adjustment];
            entrance2 = [length(this.groundStructure)-this.adjustment,1];
            entrance3 = [1+this.adjustment, length(this.groundStructure)];
            entrance4 = [length(this.groundStructure), ...
                length(this.groundStructure)-this.adjustment];
            kioskPosition = [length(this.groundStructure)/2, ...
                length(this.groundStructure)/2];
            plot(entrance1(2), entrance1(1), 'o');
            plot(entrance2(2), entrance2(1), 'o');
            plot(entrance3(2), entrance3(1), 'o');
            plot(entrance4(2), entrance4(1), 'o');
            plot(kioskPosition(2), kioskPosition(1), 'yo');

        end

        function printMaps(this)
            hold on;
            axis square
            axis off
            this.printGroundStructureMap();
            this.printPedestriansMap();
            hold off;
        end

        function deleteArrivedPedestrians(this)
            numberOfPed = length(this.pedestrians);
            numberOfErasedPed = 0;

            for i = 1:numberOfPed
                if this.pedestrians(i-numberOfErasedPed).hasArrived()
                    this.pedestrians(i-numberOfErasedPed) = [];
                    numberOfErasedPed = numberOfErasedPed + 1;
                end
            end

        end

        function step(this)
            %Updates the states
            this.currentStep = this.currentStep + 1;

            %Iterate over all pedestrians
            for i=1:length(this.pedestrians)

                %update the pedestrian
```

```
                if ~(this.pedestrians(i).hasArrived())
                    this.pedestrians(i).step();
                end

            end

            %update ground structure
            this.updateGroundStructure();

            this.deleteArrivedPedestrians();

        end

    end

end
```

A.5 Pedestrian.m

```
classdef Pedestrian < handle
    %PEDESTRIAN contains the simulation logic
    %   the function step() and subfunctions contain the logic for deciding
    %   which position should be chosen as next step


    properties
        currentPosition;
        park;
        currentDestination = 1; %index for the variable "destinations"
        arrived = false; % true after visited all destinations
        destinations = []; %matrix; lines: different destinations; first ...
            row: x-coordinates; second row: y-coordinates
        rho;%float value; used and explained in function ...
            calculateNewDirection
        lastPosition = [];
    end

    methods

        function this = Pedestrian()
            return
        end

        function setStart(this, start)
            this.currentPosition = start;
        end

```

```
        function setRho(this, rho)
            this.rho = rho;
        end

        function printValues(this)
            %For debuging
            %prints some values of the pedestrian
            disp('currentPosition =');
            disp(this.currentPosition);
            disp('currentDestination = ');
            disp(this.currentDestination);
            disp('destinations = ');
            disp(this.destinations);
            disp('arrived = ')
            disp(this.arrived)
        end

        function setPark(this, park)
            this.park = park;
        end

        function val = hasArrived(this)
            val = this.arrived;
        end

        function addDestination(this, position)
            this.destinations = [this.destinations; position];
        end

        function dest = getDestination(this)
            dest = this.destinations(this.currentDestination,:);
        end

        function vMatrix = calculateVMatrix(this)
            vMatrix = zeros(3);

            for i = -1:1
               for j = -1:1
                  if this.currentPosition(1)+i >= 1 && ...
                      this.currentPosition(2)+j >= 1 && ...
                      this.currentPosition(1)+i <= ...
                      max(size(this.park.groundStructure)) && ...
                      this.currentPosition(2)+j <= ...
                      max(size(this.park.groundStructure)) && ...
                      not(isequal(i,j,0))

                      vMatrix(i + 2, j + 2) = ...
                          this.calculateV([this.currentPosition(1)+i, ...
                          this.currentPosition(2)+j]);

```

```
                    end
                end
            end

        end

        function v = calculateV(this, position)
            %input: 2 dimensional vector
            %returns the value of V on position

            %The true size of omega woud be (2*zizeOfOmega + 1)
            sizeOfOmega = 8;

            maxSize = max(size(this.park.groundStructure));

            sum = 0;

            %check if position is inside a lake => v=-1
            if this.park.groundStructure(position(1),position(2)) < 0
                v = -1;
                return
            end

            %i is on x axis; j is on y axis
            for i = -sizeOfOmega:sizeOfOmega
                for j = -sizeOfOmega:sizeOfOmega

                    distance = max(abs(i),abs(j));

                    %check if the place where we want to calculate V is
                    %inside the park
                    if position(1)+i <= maxSize && position(1)+i >= 1 && ...
                        position(2)+j <= maxSize && position(2)+j >= 1

                        %for the case when a lake is at this position, ...
                            if a
                        %lake is at this position: set the groundStructure
                        %of a street
                        if this.park.groundStructure(position(1)+i, ...
                            position(2)+j) > 0
                            sumPart = ...
                                exp(-distance/this.park.visibility(position(1)+i, ...
                                position(2)+j))*this.park.groundStructure(position(1)+i, ...
                                position(2)+j);

                        else
                            sumPart = ...
                                exp(-distance/this.park.visibility(position(1)+i, ...
                                position(2)+j))*150;

```

```
                    end

                elseif position(1)+i > maxSize && position(2)+j > ...
                    maxSize
                    sumPart = ...
                        exp(-distance/this.park.visibility(maxSize, ...
                        maxSize))*this.park.groundStructure(maxSize, ...
                        maxSize);

                elseif position(1)+i > maxSize && position(2)+j < 1
                    sumPart = exp(-distance/this.park.visibility(1, ...
                        maxSize))*this.park.groundStructure(1, maxSize);

                elseif position(1)+i < 1 && position(2)+j < 1
                    sumPart = exp(-distance/this.park.visibility(1, ...
                        1))*this.park.groundStructure(1, 1);

                elseif position(1)+i < 1 && position(2)+j > maxSize
                    sumPart = ...
                        exp(-distance/this.park.visibility(maxSize, ...
                        1))*this.park.groundStructure(maxSize, 1);

                elseif position(1)+i < 1 && (position(2)+j <= ...
                    maxSize && position(2)+j >= 1)
                    sumPart = exp(-distance/this.park.visibility(1, ...
                        position(2)+j))*this.park.groundStructure(1, ...
                        position(2)+j);

                elseif position(2)+j < 1 && (position(1)+i >= 1 && ...
                    position(1)+i <= maxSize)
                    sumPart = ...
                        exp(-distance/this.park.visibility(position(1)+i, ...
                        1))*this.park.groundStructure(position(1)+i, 1);

                elseif position(1)+i > maxSize && (position(2)+j >= ...
                    1 &&position(2)+j <= maxSize)
                    sumPart = ...
                        exp(-distance/this.park.visibility(maxSize, ...
                        position(2)+j))*this.park.groundStructure(maxSize, ...
                        position(2)+j);

                elseif position(2)+j > maxSize && (position(1)+i >= ...
                    1 && position(1)+i <= maxSize)
                    sumPart = ...
                        exp(-distance/this.park.visibility(position(1)+i, ...
                        maxSize))*this.park.groundStructure(position(1)+i, ...
                        maxSize);

                end

```

```
                sum = sum + sumPart;

            end
        end

        v = sum/(2*sizeOfOmega + 1)^2;

    end


    function direction = calculateNewDirection(this)
        %returns the direction in a vector with norm(vector)=1
        maxVDirection = this.getMaxVDirection();

        firstPart = ...
            (this.getDestination()-this.currentPosition)/norm(this.getDestination()-this.c

        secondPart = maxVDirection/norm(maxVDirection);

        direction = this.rho*firstPart + secondPart;

        direction = direction/norm(direction);

    end

    function maxVDirection = getMaxVDirection(this)
        %returns the direction of the maximal V arround the ...
            currentPosition

        vMatrix = calculateVMatrix(this);

        maxVDirectionMatrix = [];
        onStreetMatrix = [];
        maxV = -1;

        epsilon = 0.0001;

        for i = 1:3
            for j = 1:3

                try
                    isOnStreet = ...
                        isequal(this.park.groundStructure(this.currentPosition(1)+i-2, ...
                        this.currentPosition(2)+j-2), 150);
                catch
                    isOnStreet = false;
                end

                if isOnStreet && (~isequal(i,j,2))
                    onStreetMatrix = [onStreetMatrix; [i-2, j-2]];
```

```
                    if vMatrix(i,j) >= maxV+epsilon
                        maxV = vMatrix(i,j);
                        maxVDirectionMatrix = [];
                    end

                elseif vMatrix(i,j) >= maxV-epsilon

                    %a bigger value was found, we reset count and
                    %maxVDirectionMatrix
                    if vMatrix(i,j) >= maxV+epsilon
                        maxVDirectionMatrix = [];

                    end

                    maxV = vMatrix(i,j);
                    maxVDirectionMatrix = [maxVDirectionMatrix; ...
                        [i-2, j-2]];
                end
            end
        end

        maxVDirectionMatrix = [maxVDirectionMatrix; onStreetMatrix];


        %Choose right direction if we have the same value more
        %than once in the VMatrix

        if length(maxVDirectionMatrix) >= 2
            compare = -1;
            walkDirection = ...
                (this.getDestination()-this.currentPosition)/norm(this.getDestination()-th
            walkDirection = walkDirection/norm(walkDirection);
            maxSize = size(maxVDirectionMatrix);

            for n = 1:maxSize(1,1)
                value = ...
                    maxVDirectionMatrix(n,:)/norm(maxVDirectionMatrix(n,:))*transpose(walk

                if value >= compare
                    maxVDirection = maxVDirectionMatrix(n,:);
                    compare = value;
                end
            end

        else
            maxVDirection = maxVDirectionMatrix(1,:);

        end

```

```
        end

        function setNewPosition(this)
            directionToGo = this.calculateNewDirection();

            epsilon = 0.0001;

            bound = sqrt(sqrt(2.0)+2)/2.0 + epsilon;

            move = [0,0];

            if directionToGo*[1,0]'> bound
                move = [1,0];
            elseif directionToGo*[0,1]'> bound
                move = [0,1];
            elseif directionToGo*[-1,0]'> bound
                move = [-1,0];
            elseif directionToGo*[0,-1]'> bound
                move = [0,-1];
            elseif directionToGo*([1,1]/norm([1,1]))'> bound
                move = [1,1];
            elseif directionToGo*([-1,-1]/norm([-1,-1]))'> bound
                move = [-1,-1];
            elseif directionToGo*([-1,1]/norm([-1,1]))'> bound
                move = [-1,1];
            elseif directionToGo*([1,-1]/norm([1,-1]))'> bound
                move = [1,-1];
            end

            %check if the new position would be inside a lake, we doesn't
            %allow swimming
            newPosition = this.currentPosition + move;
            if this.park.groundStructure(newPosition(1),newPosition(2)) ...
                < 0
                %calculate better newPosition

                %walkDirection = this.currentPosition-newPosition;

                directionToDestination = ...
                    (this.getDestination()-this.currentPosition)/norm(this.getDestination()-th

                if isequal(move(1),0)
                    move1= [move(2),0];
                    move2= [-move(2),0];

                    if directionToDestination * move1' > ...
                        directionToDestination * move2'
                        move = move1;
                    elseif directionToDestination * move1' < ...
                        directionToDestination * move2'
```

```
                        move = move2;
                    else
                        move = [move(2),0];
                    end

                elseif isequal(move(2),0)
                    move1 = [0,move(1)];
                    move2 = [0,-move(1)];

                    if directionToDestination * move1' > ...
                        directionToDestination * move2'
                        move = move1;
                    elseif directionToDestination * move1' < ...
                        directionToDestination * move2'
                        move = move2;
                    else
                        move = [0, move(1)];
                    end

                elseif isequal(move(1),-1) && isequal(move(2),1)

                    if ...
                        this.park.groundStructure(newPosition(1)+1,newPosition(2)) ...
                        > 0
                        move = [0,1];
                    else
                        move= [-1,0];
                    end

                elseif isequal(move(1),1) && isequal(move(2),1)

                    if ...
                        this.park.groundStructure(newPosition(1)-1,newPosition(2)) ...
                        > 0
                        move= [0,1];
                    else
                        move= [1,0];
                    end

                elseif isequal(move(1),-1) && isequal(move(2),-1)

                    if ...
                        this.park.groundStructure(newPosition(1)+1,newPosition(2)) ...
                        > 0
                        move= [0,-1];
                    else
                        move = [-1,0];
                    end

                elseif isequal(move(1),1) && isequal(move(2),-1)
```

```

                    if this.park.groundStructure(newPosition(1) − ...
                        1,newPosition(2)) > 0
                        move = [0,−1];
                    else
                        move = [1,0];
                    end
                end

            end


            this.currentPosition = this.currentPosition + move;

        end

        function step(this)

            destination = this.getDestination();

            %if pedestrian is at his destination he doesn't need to move
            %anymore (to avoid any error)
            if this.arrived || isequal(destination, this.currentPosition)
                this.arrived = 1;
                return
            end

            this.setNewPosition();

            %update currentDestination if the pedestrian reached a ...
                destination
            if isequal(this.currentPosition, destination)
                this.currentDestination = this.currentDestination + 1;
            end

            %check if pedestrian arrived at his final destination
            if this.currentDestination > size(this.destinations,1)
                this.arrived = true;
            end

        end

        function position = getPosition(this)
            position = this.currentPosition;
        end

    end

end
```