北京航空航天大學
BEIHANG UNIVERSITY

实验报告

内容（名称）：行人建模与仿真

院（系）名称	计算机学院
专业名称	计算机科学与技术专业
指导教师	宋晓
学号	17373126
姓名	刘萱

2019 年 11 月

行人模型实验报告

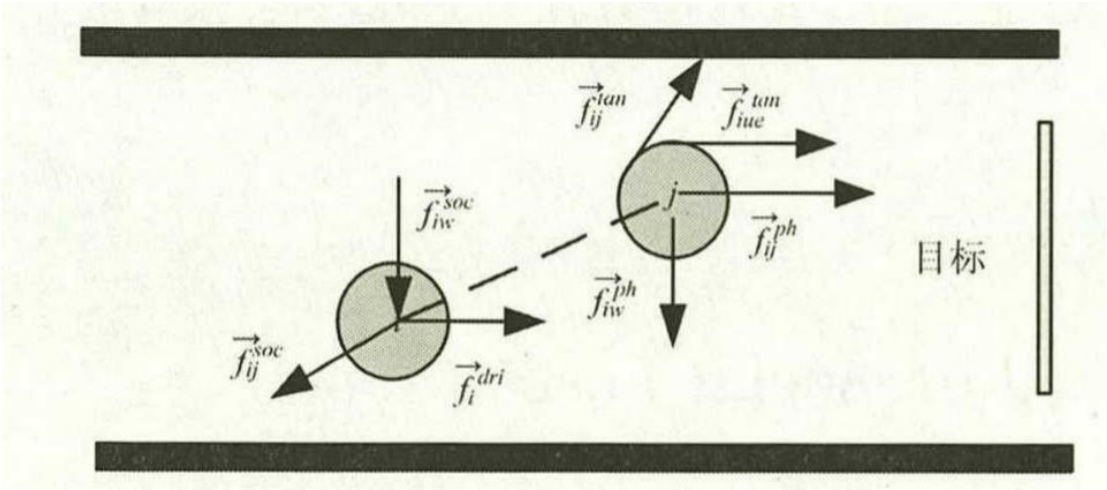
一、 实验目的

应用社会力模型，模拟一个人群疏散的过程。熟悉行人模型及其仿真过程。

二、 主要模型

1、 社会力模型：

社会力是指一个人受到周围环境的影响，从而引起自身行为的某些改变。社会力模型认为行人的运动是在社会力的作用下发生的，其中包括自驱力、行人之间的作用力、行人与周边环境之间的作用力。其模型示意图如下图所示。



图中，$\overrightarrow{f_i^{dri}}$表示行人 i 的驱动力，$\overrightarrow{f_{ij}^{soc}}$为行人 j 受到前方行人 i 的心理排斥力；$\overrightarrow{f_{ij}^{ph}}$为行人 j 与行人 i 之间接触时才会产生的物理排斥力；$\overrightarrow{f_{ij}^{tan}}$为行人 j 与行人 i 之间接触时产生的切向物理作用力；$\overrightarrow{f_{iw}^{soc}}$为行人受到周围障碍物的排斥作用而远离障碍物的心理作用；$\overrightarrow{f_{iw}^{tan}}$与$\overrightarrow{f_{iw}^{ph}}$分别为行人 i 与障碍物接触时产生的法向与切向的物理作用力。

1.1 驱动力

将行人去往目的地的主观愿望用驱动力来表示，在没有受到环境中其他阻碍的情况下，行人会主观地选择路径最短的方向与最舒适的速度。即

$$\overrightarrow{f_i^{dri}} = m_i \frac{V_e\,\vec{e}_i - \vec{v}_i}{T_\alpha}$$

式中，m_i为行人的质量；$\vec{v}_i$为行人的实际速度，一般情况下都小于期望

速度；v_e为行人的期望速度大小，只与行人的个体特征有关；T_α表示行人受到环境作用后的反应时间；$\vec{e}_i$表示行人的期望运动方向，其表达式为：

$$\vec{e}_i(t) = \frac{\vec{r}_i^k - \vec{r}_i(t)}{\left\|\vec{r}_i^k - \vec{r}_i\right\|}$$

行人的期望目的地一般不会是一个特定的点，一般为一个区域，比如一个通道、一扇门等。式中，$\vec{r}_i(t)$为在某 t 时刻行人 i 的位置；$\vec{r}_i^k$为形成目的地区域的一系列点，一般取离行人最近的点。

1.2 行人之间的相互作用力

行人在运动过程中总会尽可能与他人保持一定的距离，若距离过小，则会引起行人心理上的排斥感，这就是常说的“领域效应”，正是因为这种“领域效应”的作用，人与人之间尤其是陌生人之间总会存在无形的排斥力。即

$$\vec{f}_{ij}^{SOC} = Aexp(\frac{r_{ij} - d_{ij}}{B})\left[\lambda_i + (1-\lambda_i)\frac{1+\cos(\varphi_{ij})}{2}\right]\vec{n}_{ij}$$

式中，$\vec{f}_{ij}^{SOC}$为行人 i 受到行人 j 的心理排斥力，当行人 j 在行人 i 的作用区域时，行人 i 会受到行人 j 的作用；A 为行人之间的作用强度，为模型参数；B 为行人之间的作用力范围，为模型参数；$r_{ij} - d_{ij}$为行人之间距离的相反数，r_{ij}是两行人的半径之和，d_{ij}两行人中心的距离；λ_i为各向异性参数，考虑了不同方向的行人对当前行人不同的影响程度，一般认为前方行人影响大于后方行人，λ_i越小表示前方行人影响越大，$\lambda_i \in [0,1]$；φ_{ij}为行人间的斥力与期望运动方向的夹角；$\vec{n}_{ij}$为行人 j 指向行人 i 的单位向量。

心理排斥力是当行人 j 进入行人 i 的作用区域后会产生的，远离区域时认为作用为 0，当行人距离靠近时，排斥力的大小是指数形式增长，当行人发生接触时，除了心理排斥作用，还有物理作用力，包括沿着垂直行人接触面的方向作用，使行人抵抗接触，沿着平行于行人接触面的方向使行人快速分离。法向力和切向力分别为

$$\vec{f}_{ij}^{ph} = \left(K\Theta\left(r_{ij} - d_{ij}\right)\right)\vec{n}_{ij}$$

$$\vec{f}_{ij}^{tan} = K\Theta\left(r_{ij} - d_{ij}\right)\triangle\vec{v}_{ij}\vec{t}_{ij}$$

其中，$\Theta = \begin{cases} r_{ij} - d_{ij}, & r_{ij} - d_{ij} \le 0 \\ 0, & r_{ij} - d_{ij} > 0 \end{cases}$

式中，$\vec{n}_{ij}$为行人 j 指向行人 i 的单位向量；$\vec{t}_{ij}$由单位法向量$\vec{n}_{ij}$逆时针 旋转 90° 所得到的；K 为人体弹性系数（N/m）；k 为人体相对速度差摩擦系数（N·（s/m²））；$\triangle \vec{v}_{ij}$表示两行人速度的矢量差。

1.3 行人受到障碍物作用

当行人作用区域范围内有其他障碍物时，行人会自主选择路径来避免与障碍物发生接触或碰撞，行人与障碍物作用原理与行人之间相近。由于障碍物的不可感知，行人的作用具有双向性，而与障碍物的作用是单向的。作用力仍然包括心理排斥力与接触时的物理作用力，但其中参数取值有所变化。

$$\vec{f}_{iw}^{soc} = A_w exp(\frac{r_i - d_{iw}}{B_w})\vec{n}_{iw}$$

式中，A_w为行人与障碍物作用力强度，为模型参数；B_w为行人与障碍物作用范围，为模型参数；$r_i - d_{iw}$为行人半径与行人到障碍物的法向距离差；$\vec{n}_{iw}$为由障碍物指向行人的单位法向量。

当行人速度过大躲避不及而与障碍物相接触时，对障碍物产生挤压作用。

$$\vec{f}_{ij}^{ph} = \left(K\Theta(r_i - d_{iw})\right)\vec{n}_{iw}$$

$$\vec{f}_{iw}^{tan} = k\Theta\left(r_{ij} - d_{ij}\right) < \vec{v}_i\vec{t}_{iw} > \vec{n}_{iw}$$

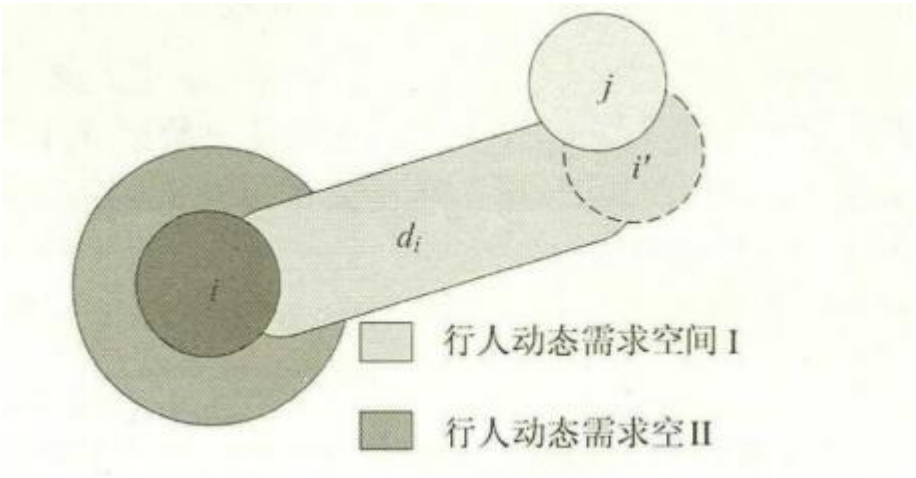
式中，$\vec{n}_{iw}$为由障碍物指向行人的单位法向量；$\vec{t}_{iw}$为平行于障碍物的单位切向力；$< \vec{v}_i\vec{t}_{iw} >$为行人速度在障碍物方向上的投影；K 为人体正压力弹性系数（N/m）；k 为人体相对速度差摩擦系数（N·（s/m²））。

其中，$\Theta = \begin{cases} r_i - d_{iw}, & r_i - d_{iw} \le 0 \\ 0, & r_i - d_{iw} > 0 \end{cases}$

2、 社会力模型的改进

虽然社会力模型能够形象地表达行人之间的心理排斥作用，社会力模型中的避让行为主要是通过停止、绕行来实现的，但在模拟中总会不可避免地出现碰撞甚至穿越的行为。为了更好地解决社会力模型中存在的行人碰撞，引入减速避让机制和自停止机制。

下图为行人动态与静态需求空间示意图。



减速避让机制包含了行人的预判行为，行人按照原来的速度行走至下一步长所需要的空间为动态需求空间$d_{i1} = (a + b|\vec{v}_i(t)|$，行人速度为 0，需要$d_{i2} =$（$a + b \times 0$)大小的步行距离。若发现下一步长中其他行人占据了动态需求空间d_{i1}与d_{i2}，则在本步长内行人会受到让其减速的避让力：

$$\vec{f}_i^{avo} = -\delta_i(t)\,\vec{v}_i(t) m_i [\lambda_i + (1-\lambda_i)\frac{1+\cos(\varphi_{ij})}{2}]\,\vec{n}_{ij}$$

$d_{i1} = (a + b|\vec{v}_i(t)|$为行人 i 按原来速度运动后与行人当前的距离；$-\delta_i(t)\vec{v}_i(t) = -\frac{\delta_i(t)}{T_r}$为行人 i 由速度$\vec{v}_i(t)$减至 0 而采取的加速度；$T_r$为行人反应时间；$\varphi_{ij}$为行人实际步行速度与行人 j 作用对行人 i 的排斥力的反方向的夹角；λ_i为各向异性参数，考虑了不同方向的行人对当前行人不同的影响程度，当取 0 时，说明后方行人对前方行人无影响。

行人所受力的和均满足力的叠加原理，仿真过程中只考虑离行人最近的边界或障碍物对行人的作用力。当行人在某一时刻的所有受力确定后，根据牛顿第二定律确定行人的下一步运动参数。

$$F_{i合} = f_{驱动力} + f_{行人作用力} + f_{障碍物作用力} + f_{减速避让力}$$

$$\begin{cases} a = \dfrac{F_{i合}}{m_i} \\ v_i(t) = v_i(t-1) + a\Delta t \\ x_i(t) = x_i(t-1) + v_i(t-1)\Delta t + \dfrac{1}{2}a\Delta t^2 \end{cases}$$

实际中，行人躲避碰撞通常有两种反应：一是减速直至停止，来防止与

前方行人或物体发生碰撞；二是改变行进方向，采取绕行的方式以一定的速度通过。为了避免重叠，加人了以下规则：

（1）行人的初始速度为行人的期望速度，行人期望速度在整个仿真过程中保持不变。本文认为行人实际速度只能小于等于期望速度。因而当行人的实际速度大于行人期望速度时，取行人最大速度为当前期望速度的值。

（2）只有其他行人进人当前行人的作用域时，才会有社会力的作用，行人受到的作用力与距离有关，距离越近，社会力也越大，只考虑前方行人的作用，对后方行人的作用认为等于 0。

（3）如果行人与前方行人接触受到较大的力，而产生反向的速度，令当前速度为 0，认为在运动过程中行人不会产生后退。

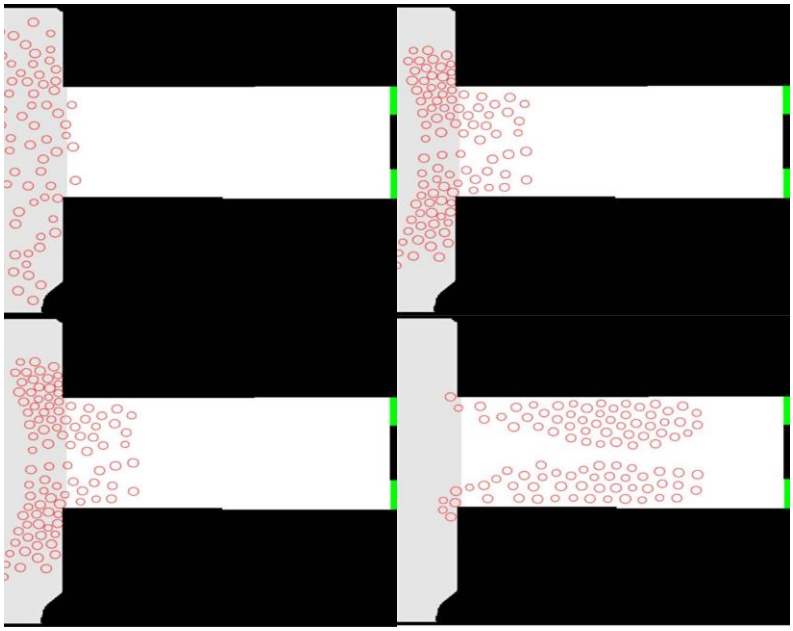
（4）在行人与障碍物的作用中，行人始终位于仿真作用区域内，判断当前时刻行人与障碍物的距离与行人下一时刻可能的位移，若位移超出了作用区域，则假定行人位于边界上，但不可超出和穿过边界。

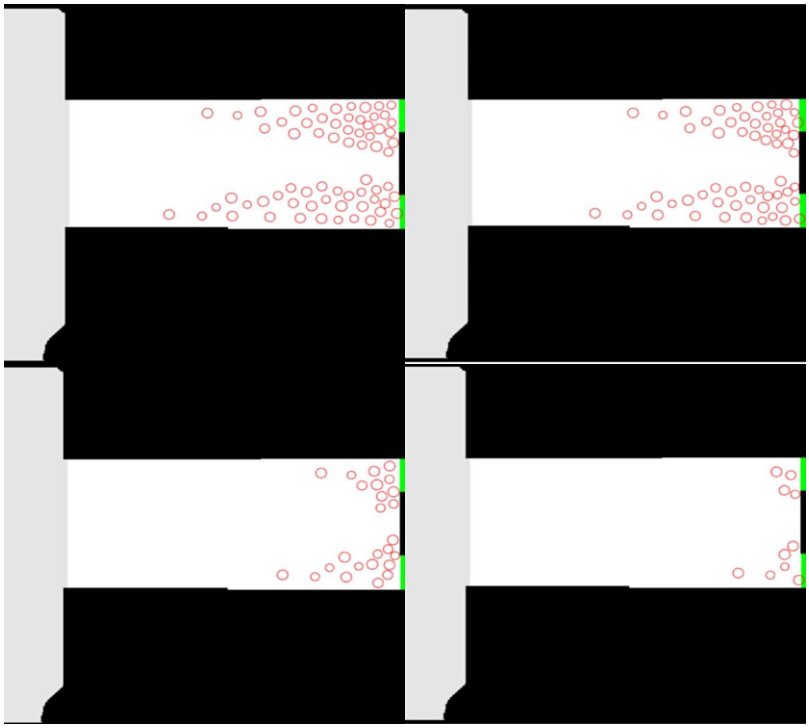
三、 编程实现与调试过程（需要给出代码实现的主要函数及其对应的数学模型）

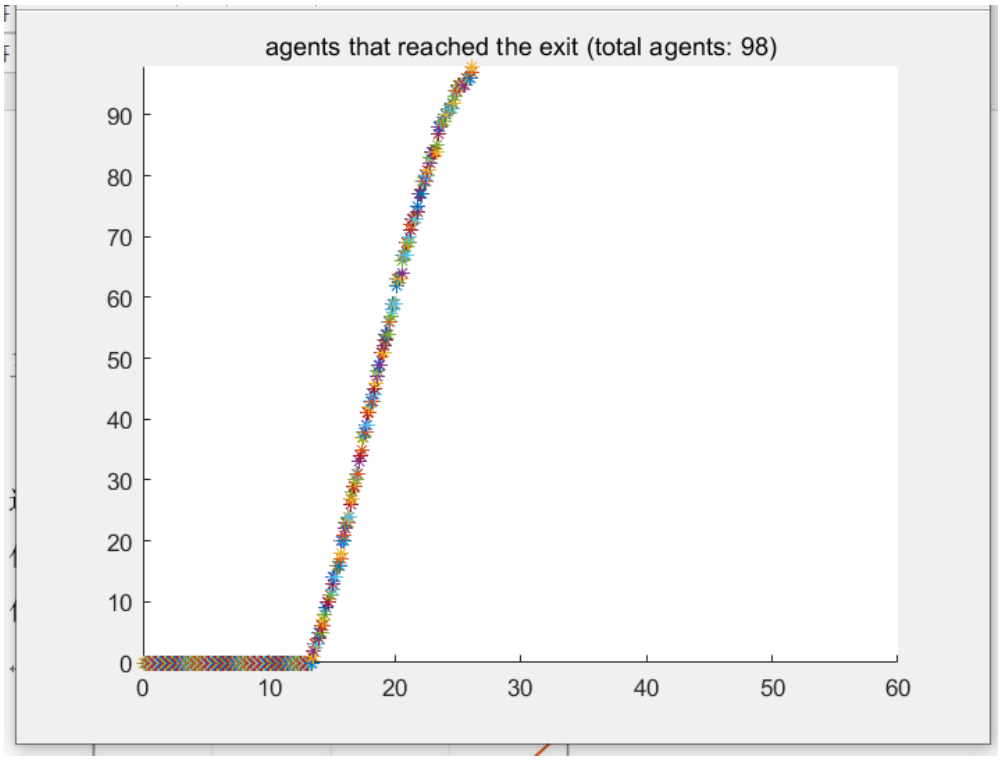
运用 MATLAB 工具建立社会力模型的仿真系统。

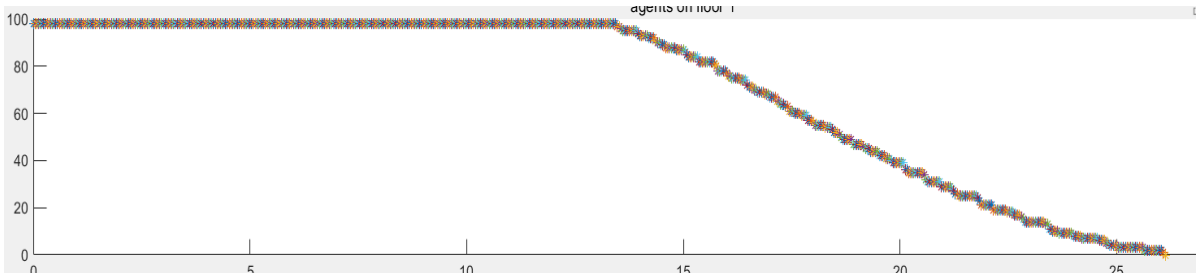
代码见附录。

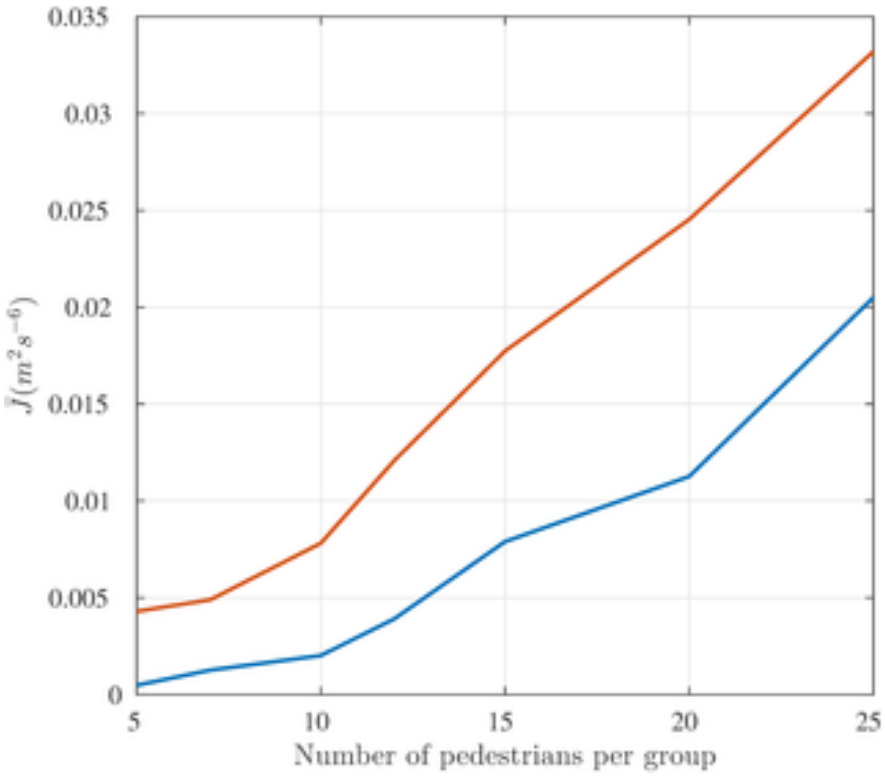
仿真结果：






agents that reached the exit (total agents: 98)
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60


100
80
60
40
20
0
0
5
10
15
20
25



五、 主要代码

```
function simulate(config_file)
% run this to start the simulation

if nargin==0
    config_file='../data/config2.conf';
end

fprintf('Load config file...\n');
config = loadConfig(config_file);

data = initialize(config);


data.time = 0;
frame = 0;
fprintf('Start simulation...\n');

while (data.time < data.duration)
    tstart=tic;
    %加载需要数据
    data = addDesiredForce(data);
    %添加城墙数据
    data = addWallForce(data);
```

```
    %添加行人之间相互作用力
    data = addAgentRepulsiveForce(data);
    %提供行人移动数据
    data = applyForcesAndMove(data);
    %data
    % do the plotting
    set(0,'CurrentFigure',data.figure_floors);
    %data.floor_count
    for floor=1:data.floor_count
        %统计行人在房间数据
        plotAgentsPerFloor(data, floor);
        %画行人
        plotFloor(data, floor);
    end
    if data.save_frames==1
        print('-depsc2',sprintf('frames/%s_%04i.eps', ...
            data.frame_basename,frame), data.figure_floors);
    end

    set(0,'CurrentFigure',data.figure_exit);
    %统计行人到达出口数据
    plotExitedAgents(data);


    % print mean/median velocity of agents on each floor
%     for fi = 1:data.floor_count
%    avgv = arrayfun(@(agent) norm(agent.v), data.floor(fi).agents);
%   fprintf('Mean/median velocity on floor %i: %g/%g m/s\n', fi, mean
(avgv), median(avgv));
%     end


    if (data.time + data.dt > data.duration)
        data.dt = data.duration - data.time;
        data.time = data.duration;
    else
        data.time = data.time + data.dt;
    end

    if data.agents_exited == data.total_agent_count
        fprintf('All agents are now saved (or are they?). Time: %.2f s
ec\n', data.time);
        fprintf('Total Agents: %i\n', data.total_agent_count);
```

```
        print('-depsc2',sprintf('frames/exited_agents_%s.eps', ...
            data.frame_basename), data.figure_floors);
        break;
    end


    telapsed = toc(tstart);
    pause(max(data.dt - telapsed, 0.01));
    fprintf('Frame %i done (took %.3fs; %.3fs out of %.3gs simulate
d).\n', frame, telapsed, data.time, data.duration);
    frame = frame + 1;
end

fprintf('Simulation done.\n');


function data = addAgentRepulsiveForce(data)
%ADDAGENTREPULSIVEFORCE Summary of this function goes here
%   Detailed explanation goes here

% Obstruction effects in case of physical interaction

% get maximum agent distance for which we calculate force
r_max = data.r_influence;
tree = 0;

for fi = 1:data.floor_count
    pos = [arrayfun(@(a) a.p(1), data.floor(fi).agents);
           arrayfun(@(a) a.p(2), data.floor(fi).agents)];

    % update range tree of lower floor
    tree_lower = tree;

    agents_on_floor = length(data.floor(fi).agents);

    % init range tree of current floor
    if agents_on_floor > 0
        tree = createRangeTree(pos);
    end

    for ai = 1:agents_on_floor
        pi = data.floor(fi).agents(ai).p;
        vi = data.floor(fi).agents(ai).v;
        ri = data.floor(fi).agents(ai).r;
```

```
        % use range tree to get the indices of all agents near agent a
i
        idx = rangeQuery(tree, pi(1) - r_max, pi(1) + r_max, ...
                                 pi(2) - r_max, pi(2) + r_max)';

        % loop over agents near agent ai
        for aj = idx

            % if force has not been calculated yet...
            if aj > ai
                pj = data.floor(fi).agents(aj).p;
                vj = data.floor(fi).agents(aj).v;
                rj = data.floor(fi).agents(aj).r;

                % vector pointing from j to i
                nij = (pi - pj) * data.meter_per_pixel;

                % distance of agents
                d = norm(nij);

                % normalized vector pointing from j to i
                nij = nij / d;
                % tangential direction
                tij = [-nij(2), nij(1)];

                % sum of radii
                rij = (ri + rj);

                % repulsive interaction forces
                if d < rij
                   T1 = data.k*(rij - d);
                   T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
                else
                   T1 = 0;
                   T2 = 0;
                end

                F =  (data.A * exp((rij - d)/data.B) + T1)*nij + T2;

                data.floor(fi).agents(ai).f = ...
                    data.floor(fi).agents(ai).f + F;
                data.floor(fi).agents(aj).f = ...
                    data.floor(fi).agents(aj).f - F;
            end
```

```
        end

        % include agents on stairs!
        if fi > 1
            % use range tree to get the indices of all agents near age
nt ai
            if ~isempty(data.floor(fi-1).agents)
                idx = rangeQuery(tree_lower, pi(1) - r_max, ...
                       pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';
                % if there are any agents...
                if ~isempty(idx)
                    for aj = idx
                        pj = data.floor(fi-1).agents(aj).p;
                        if data.floor(fi-1).img_stairs_up(round(pj
(1)), round(pj(2)))

                            vj = data.floor(fi-1).agents(aj).v;
                            rj = data.floor(fi-1).agents(aj).r;

                            % vector pointing from j to i
                            nij = (pi - pj) * data.meter_per_pixel;

                            % distance of agents
                            d = norm(nij);

                            % normalized vector pointing from j to i
                            nij = nij / d;
                            % tangential direction
                            tij = [-nij(2), nij(1)];

                            % sum of radii
                            rij = (ri + rj);

                            % repulsive interaction forces
                            if d < rij
                               T1 = data.k*(rij - d);
                               T2 = data.kappa*(rij - d)*dot((vj - vi),t
ij)*tij;
                            else
                               T1 = 0;
                               T2 = 0;
                            end
                            F = (data.A * exp((rij - d)/data.B) + T1)*n
ij + T2;
```

```
                        data.floor(fi).agents(ai).f = ...
                            data.floor(fi).agents(ai).f + F;
                        data.floor(fi-1).agents(aj).f = ...
                            data.floor(fi-1).agents(aj).f - F;
                    end
                end
            end
        end
    end
end

function data = addDesiredForce(data)
%ADDDESIREDFORCE add 'desired' force contribution (towards nearest ex
it or
%staircase)

for fi = 1:data.floor_count

    for ai=1:length(data.floor(fi).agents)

        % get agent's data
        p = data.floor(fi).agents(ai).p;
        m = data.floor(fi).agents(ai).m;
        v0 = data.floor(fi).agents(ai).v0;
        v = data.floor(fi).agents(ai).v;


        % get direction towards nearest exit
        ex = lerp2(data.floor(fi).img_dir_x, p(1), p(2));
        ey = lerp2(data.floor(fi).img_dir_y, p(1), p(2));
        e = [ex ey];

        % get force
        Fi = m * (v0*e - v)/data.tau;

        % add force
        data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + F
i;
    end
end

      function data = addWallForce(data)
      %ADDWALLFORCE adds wall's force contribution to each agent
```

```
        for fi = 1:data.floor_count
            for ai=1:length(data.floor(fi).agents)
                % get agents data
                p = data.floor(fi).agents(ai).p;
                ri = data.floor(fi).agents(ai).r;
                vi = data.floor(fi).agents(ai).v;
                % get direction from nearest wall to agent
                nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p
        (2));
                ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p
        (2));

                % get distance to nearest wall
                diW = lerp2(data.floor(fi).img_wall_dist, p(1), p(2));

                % get perpendicular and tangential unit vectors
                niW = [ nx ny];
                tiW = [-ny nx];
                % calculate force
                if diW < ri
                    T1 = data.k * (ri - diW);
                    T2 = data.kappa * (ri - diW) * dot(vi, tiW) * tiW;
                else
                    T1 = 0;
                    T2 = 0;
                end
                Fi = (data.A * exp((ri-diW)/data.B) + T1)*niW - T2;
                % add force to agent's current force
                data.floor(fi).agents(ai).f = data.floor(fi).agents(a
        i).f + Fi;
            end
        end
```

```
function data = applyForcesAndMove(data)
%APPLYFORCESANDMOVE apply current forces to agents and move them usin
g
%the timestep and current velocity

n_velocity_clamps = 0;

% loop over all floors
for fi = 1:data.floor_count

    % init logical arrays to indicate agents that change the floor or
exit
```

```
    % the simulation
    floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);

    % loop over all agents
    for ai=1:length(data.floor(fi).agents)
        % add current force contributions to velocity
        v = data.floor(fi).agents(ai).v + data.dt * ...
            data.floor(fi).agents(ai).f  / data.floor(fi).agents(ai).
m;

        % clamp velocity
        if norm(v) > data.v_max
            v = v / norm(v) * data.v_max;
            n_velocity_clamps = n_velocity_clamps + 1;
        end

        % get agent's new position
        newp = data.floor(fi).agents(ai).p + ...
               v * data.dt / data.meter_per_pixel;

        % if the new position is inside a wall, remove perpendicular
        % component of the agent's velocity
        if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) <
 ...
                data.floor(fi).agents(ai).r

            % get agent's position
            p = data.floor(fi).agents(ai).p;

            % get wall distance gradient (which is off course perpendi
cular
            % to the nearest wall)
            nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p
(2));
            ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p
(2));
            n = [nx ny];
            % project out perpendicular component of velocity vector
            v = v - dot(n,v)/dot(n,n)*n;
            % get agent's new position
            newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
        end
```

```
        if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
            newp = data.floor(fi).agents(ai).p;
            v = [0 0];
        end

        % update agent's velocity and position
        data.floor(fi).agents(ai).v = v;
        data.floor(fi).agents(ai).p = newp;

        % reset forces for next timestep
        data.floor(fi).agents(ai).f = [0 0];

        % check if agent reached a staircase and indicate floor change
        if data.floor(fi).img_stairs_down(round(newp(1)), round(newp
(2)))
            floorchange(ai) = 1;
        end

        % check if agent reached an exit
        if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;
        end
    end
    % add appropriate agents to next lower floor
    if fi > 1
        data.floor(fi-1).agents = [data.floor(fi-1).agents ...
                                   data.floor(fi).agents(floorchange)];
    end
    % delete these and exited agents
    data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exite
d));
end
if n_velocity_clamps > 0
    fprintf(['WARNING: clamped velocity of %d agents, ' ...
           'possible simulation instability.\n'], n_velocity_clamps);
end

      function val = checkForIntersection(data, floor_idx, agent_idx)
      % check an agent for an intersection with another agent or a wa
      ll
      % the check is kept as simple as possible
      %
      %  arguments:
```

```
%   data            global data structure
%   floor_idx       which floor to check
%   agent_idx       which agent on that floor
%   agent_new_pos   vector: [x,y], desired agent position to ch
eck
%
%  return:
%   0               for no intersection
%   1               has an intersection with wall
%   2                                   with another agent
val = 0;
p = data.floor(floor_idx).agents(agent_idx).p;
r = data.floor(floor_idx).agents(agent_idx).r;

% check for agent intersection
for i=1:length(data.floor(floor_idx).agents)
    if i~=agent_idx
        if norm(data.floor(floor_idx).agents(i).p-p)*data.meter
_per_pixel ...
                <= r +  data.floor(floor_idx).agents(i).r
            val=2;
            return;
        end
    end
end
% check for wall intersection
if lerp2(data.floor(floor_idx).img_wall_dist, p(1), p(2)) < r
    val = 1;
end
```

六、 主要参考文献

[1]. Helbing D, Farkas I, Vicsek T. Simulating dynamical features of escape panic[J]. Nature, 2000, 407(6803):487-90.

[2]. Schadschneider, A., Klingsch, W., et al., Evacuation dynamics: empirical results, modeling and applications. In: Encyclopedia of Complexity and System Science, 2008, 517–550.

[3]. F. Farina , D. Fontanelli , A. Garulli , Walking ahead: the headed social force model, PLoS One 12 (1) (2017) e0169734 .

[4]. 朱前坤, 南娜娜, 惠晓丽, & 杜永峰. (0). 基于社会力模型的人群运动仿真模拟. 第 19 届中国系统仿真技术及其应用学术年会.