

****

实验报告

内容：社会力模型模拟人群的对流和疏散

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

2018年12月

**社会力模型实验报告**

1. **绪言**

人群疏散是安全领域的一项重要内容，合理有效的逃生策略及疏散措施可以减少伤亡、挽救生命。与车辆交通相比，行人交通的动力学特性更加复杂。至今为止，学者们从不同角度出发，提出了多种行人运动模型，总体上可以分为宏观模型和微观模型。宏观模型，主要研究人群流量等宏观特性，而忽略单个个体的运动状态，比如利用流体力学模拟人流。近年来，随着计算机技术以及相关理论的发展，研究热点逐渐转向微观模型。与宏观模型不同，微观模型着眼于个体行为的建模，宏观群体特征视为个体间相互作用的涌现。常用的微观模型有元胞自动机模型、格子气模型、社会力模型、基于多智能体的模型等。

社会力模型是常用的连续微观模型，它由Helbing等学者在社会力概念的基础上结合牛顿经典力学而提出。在模型中，社会心理等各种影响行人运动的因素被量化成具体的力，行人的运动由经典牛顿力学规律决定。例如，一个质量70kg的行人想要以1的加速度追赶正前方的同伴，这种心理作用可以等效为行人受到指向正前方的、大小为70N的作用力。作为连续模型，社会力模型具有元胞自动机等离散模型不具有的一些优点，比如行人可以朝任意方向移动、紧急情况下行人之间可以产生不同程度的重叠等，这使得社会力模型仿真出来的行人更接近现实。另外，社会力模型可以模拟仿真出自动渠化、瓶颈摆动、欲速则不达等多种行人自组织现象。

虽然社会力模型具有众多优点，但仍存在一些局限性，尤其在仿真人群恐慌、行人密度过大或过小等情况时经常产生一些不合实际的现象。例如，原始社会力模型中，行人的弹性系数取得非常大，这意味着两个行人之间若有5cm的重叠，将会产生6000N的压力。这么大的力，即使不考虑人的承受能力，也会使人产生巨大的加速度以及随后产生的速度跳变、位置跳变等一系列不符合实际的现象。

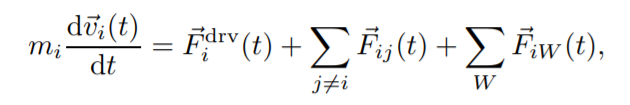
在上文所述的社会力模型的优点和缺点的基础之上，我们进行了本次实验。

1. **实验目的**

编程实现社会力模型，并对其中的细节进行改进，模拟人群的对流和疏散场景。通过GUI，将人群的运动过程直观地展现出来，验证社会力模型仿真人群运动的准确性。

1. **数学模型**
2. **社会力模型**

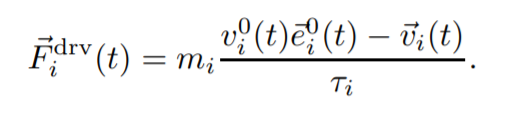
在社会力模型中，行人用一个个具有一定质量和半径的圆来表示。行人的运动，按照其所受各种力的合力，依据牛顿第二定律来决定



其中，、分别为行人的质量和速度，*t*为时间。模型中，行人总共受到三种力的作用，分别为行人的自驱动力、其他行人对行人的作用力以及墙或障碍物对行人的作用力.

1. **自驱动力**

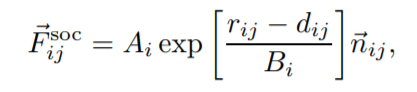
对于行人，其期望在时间内由目前的实际速度到达期望速度. 其中为期望速度大小，单位向量为期望速度方向。这种作用可以用以下自驱动力的形式来表示



在本次实验中，即期望速度方向是由算法给出的。

1. **行人之间相互作用力**

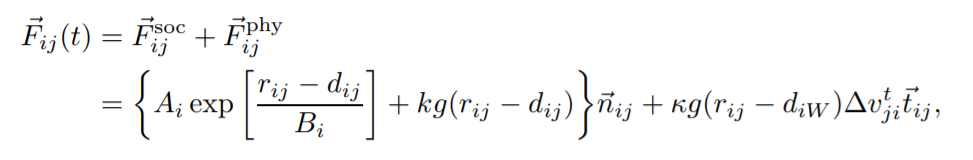
行人不仅受到自身驱动力的作用，还受到其他行人的影响。相关研究显示，行人期望与其他行人之间保持一定距离，这种心理作用可以通过排斥力来实现。在社会力模型中，采用以下形式



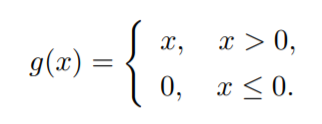
其中，为行人的半径与行人的半径的和，为行人的圆心与行人的圆心之间的距离，单位向量由指向.

当行人与行人发生身体接触时，会产生物理作用力。对的物理作用力记为，由两部分组成，一是身体挤压造成的弹力，一是相对运动产生的滑动摩擦力，其中为切方向，为相对速度在切方向上的投影。参数,.

综上，行人对行人的作用力可表示为

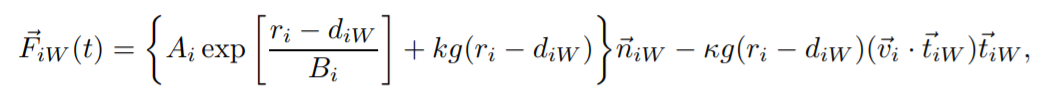


其中



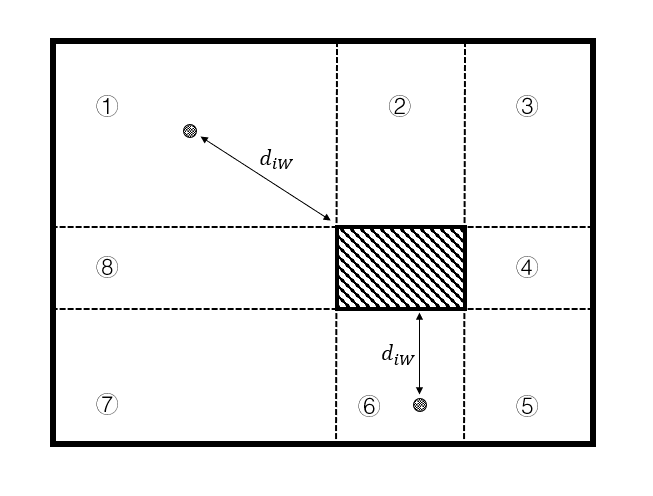
1. **行人与障碍物之间的作用力**

与行人之间的作用力类似，墙或障碍物对行人的作用力表示成以下形式



其中为行人圆心到的距离，和为障碍物相应的法向量和切向量。

在原始的社会力模型中，行人圆心到的距离这一概念并没有被准确定义。尤其是当障碍物在场景地图中占据一定面积的时候，对于行人而言，能够感知到的只有障碍物与自己最接近的那一部分，而无法感知到障碍物的全貌。因此，如果对障碍物做分割逐块求作用力，或取障碍物的质心算距离，在社会力模型中都是不符合实际的。因此，我们在本实验中把墙或障碍物与行人之间的距离定义为行人与障碍物的边界的最短距离。下图显示了这种距离计算方式的具体细节



不失一般性，我们假设墙和障碍物都是矩形。四周的围墙到行人距离的计算十分简单，取垂直距离即可，上图中暂且略去。重点在于行人到障碍物距离的计算。以障碍物的四条边为界线，对地图区域进行划分，将矩形地图分为8个小矩形。当行人处于①③⑤⑦这四个区域中时，按照其与障碍物最近的一个角计算距离，如上图区域①所示；当行人处于②④⑥⑧这四个区域中时，按照其与障碍物最近的一条边的垂直距离计算距离.这种计算距离的方式很好地模拟了行人对场景中障碍物的感知，相比于逐点或逐边计算，减轻了模型给计算机带来的计算压力。

搜索算法是一种在静态路网中求解最短路径最有效的直接搜索方法，也是许多其它问题的常用启发式算法。启发式搜索算法的关键，在于从当前搜索结点往下选择下一步结点时，可以通过一个启发函数来进行选择，选择代价最少的结点作为下一步搜索的结点。常见的用于求解最短路径的DFS和BFS算法在展开子结点时，均属于盲目型搜索，也就是说它们不会选择哪个结点在下一次搜索中是更有解，而是直接跳转到某结点进行下一步的搜索。在最坏情形下，需要试探完整的解集空间才可以获得最短路径解。因此，这两种算法只适用于问题规模不大的搜索问题。

与传统的DFS和BFS不同的是，搜索算法可以通过一个经过仔细设计的启发函数，在很短的时间内得到一个搜索问题的最优解。在社会力模型中，有大量的行人需要计算目标方向，因此采用搜索算法是一个合适的选择。

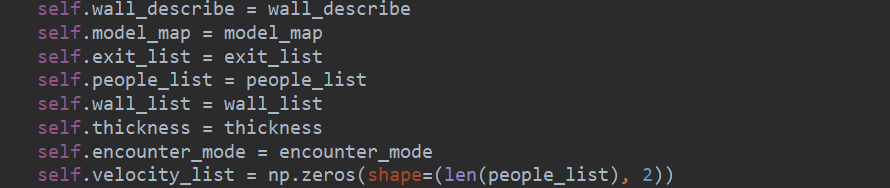
搜索算法的核心在于如何设计一个好的启发函数。启发函数的表达形式一般如其中表示从起点到搜索点的代价，表示从搜索点到目标点的代价。设计的好坏，直接影响到该搜索算法的效率。

搜索算法的流程如下：

1. 为算法设置两个额外的存储空间：OPEN表和CLOSE表。
2. 初始化OPEN和CLOSE表，将开始结点（开始结点的H和G值都视为0）放入OPEN表中。
3. 重复下面的步骤：
   1. 在开始列表中查找具有最小F值的结点，并把查找到的结点作为当前结点
   2. 把当前结点从OPEN列表中删除，并加入到CLOSE列表中
   3. 对当前结点相邻的每一个结点依次执行以下步骤：
      1. 如果相邻结点不可通行，或者该结点已在CLOSE列表中，则什么操作也不执行
      2. 如果该相邻结点不在OPEN列表中，则将该结点添加到OPEN列表中，并将该相邻结点的父结点设置为当前结点，同时计算保存相邻结点的F值
      3. 如果该相邻结点在OPEN表中，则判断若经由当前结点到达该相邻结点的F值是否小于原来保存的F值，若小于，则将该相邻结点的父结点设为当前结点，并重新设置该相邻结点的F值
   4. 若当终点结点被加入到OPEN列表作为待检验结点时，表示路径已找到，此时应终止循环；若当前OPEN列表为空，则表示没有从开始结点到终点结点的路径，此时循环结束
4. 循环终止后，从终点结点开始沿着父结点向前遍历，从后向前输出搜索的路径
5. **编程实现与调试过程**
6. **编程实现**
7. 定义GUI与社会力计算模块的接口，对GUI和社会力模型分别编程实现

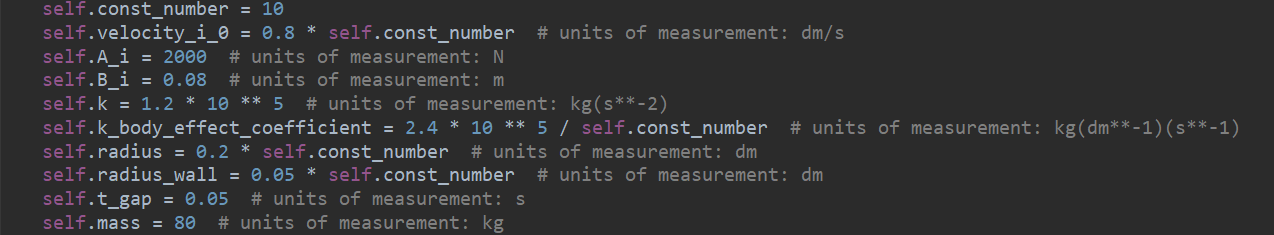
**（此处添加接口定义截图，可选）**

1. 使用python的numpy.array作为保存墙、障碍物及行人所在坐标的容器，设置三者的坐标，对场景地图进行初始化
2. 社会力计算模块重要变量定义：



* self.wall\_describe = wall\_describe
* 类型：np.array
* 形状：[x, 4]
* 作用：列表记录障碍物的位置，里面有x个wall，每个wall，是个四维数组，分别代表障碍物的起始横坐标，起始横坐标，起始纵坐标，起始纵坐标。
* self.model\_map = model\_map
* 类型：np.array
* 形状：[a,b]
* 作用：地图，a\*b的矩阵地图，单位为分米，对应位置的点的数字代表可通过性。例如，model\_map[1][1]==1，代表这个点为墙或者障碍物，不可通过，反之代表可以通过的自由区域
* self.exit\_list = exit\_list
* 类型：np.array
* 形状：[x, 2]
* 作用：出口列表，里面有x个出口的二维坐标，多个坐标组成一个块状出口。主要用于A星算法确定方向。
* self.people\_list = people\_list
* 类型：np.array
* 形状：[x, 2]
* 类型：dtype=np.double
* 作用：人的坐标的列表，可以定位在非整数的坐标。随着时间增长，如果有人到达出口，队列会减短。用于定位人的坐标
* self.wall\_list = wall\_list
* 类型：np.array
* 形状：[x, 2]
* 作用：墙和障碍物列表，里面有x个障碍和墙的二维坐标，储存了所有的 model\_map上的值为1的坐标，用于A星算法确定方向，以及人墙力量的计算。
* self.thickness = thickness
* 类型：int
* 作用：定义墙的厚度，用于计算人和墙的位置关系从而计算人和墙的作用力。
* self.encounter\_mode = encounter\_mode
* 类型：boolean
* 作用：用于决定所进行的过程是对流情况还是逃生情况，当值为True时，为对流，反之为逃生。
* self.velocity\_list = np.zeros(shape=(len(people\_list), 2))
* 类型：np.array
* 形状：[x, 2]
* 类型：dtype=np.double
* 作用：速度向量列表，与people\_list一一对应，代表对应位置的人的速度。随着时间增长，如果有人到达出口，队列会减短。同时会根据加速度进行调整。
* self.map\_height, self.map\_width = self.model\_map.shape
* 类型：int, int
* 作用：定义map的长和宽，分别是位置坐标第一个值得上限和第二个值得坐标上限。

1. 社会力计算模块重要常量定义：



* self.const\_number = 10
* 含义：定义倍率，我们的模型使用的是分米，为以正常量纲进行计算对一下常量和变量需要进行转换。
* 量纲：无
* self.velocity\_i\_0 = 0.8 \* self.const\_number
* 含义：每个人期望速度得模
* 量纲：dm/s
* self.A\_i = 2000
* 量纲：N
* self.B\_i = 0.08
* 量纲：m
* self.k = 1.2 \* 10 \*\* 5
* 量纲：kg(s\*\*-2)
* self.k\_body\_effect\_coefficient = 2.4 \* 10 \*\* 5 / self.const\_number
* 含义：人体影响系数
* 量纲：kg(dm\*\*-1)(s\*\*-1)
* self.radius = 0.2 \* self.const\_number
* 量纲：dm
* self.radius\_wall = 0.05 \* self.const\_number
* 含义：墙的半径
* 量纲：dm
* self.t\_gap = 0.05
* 含义：时间间隔
* 量纲：s
* self.mass = 80
* 含义：人的质量
* 量纲：kg

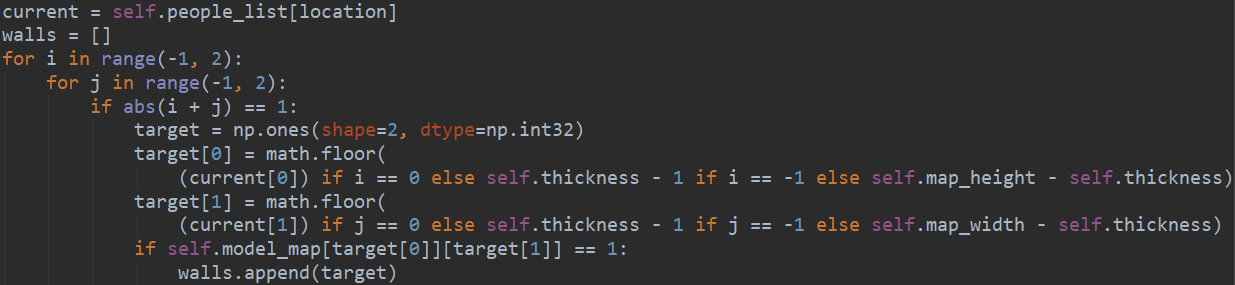
1. 定义各种辅助函数，如计算两个行人间距离的函数、计算人和墙间距离的函数，以及用于计算合力的函数

计算两个行人间以及人和墙间距离的函数：

所使用的距离是欧拉距离。

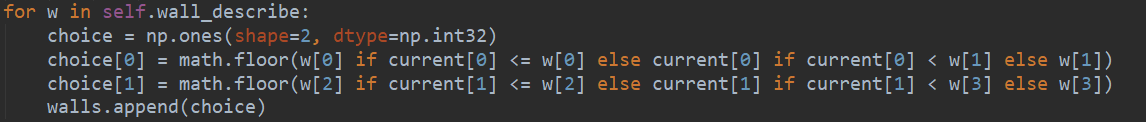


获得当前位置到四面墙的距离的函数：



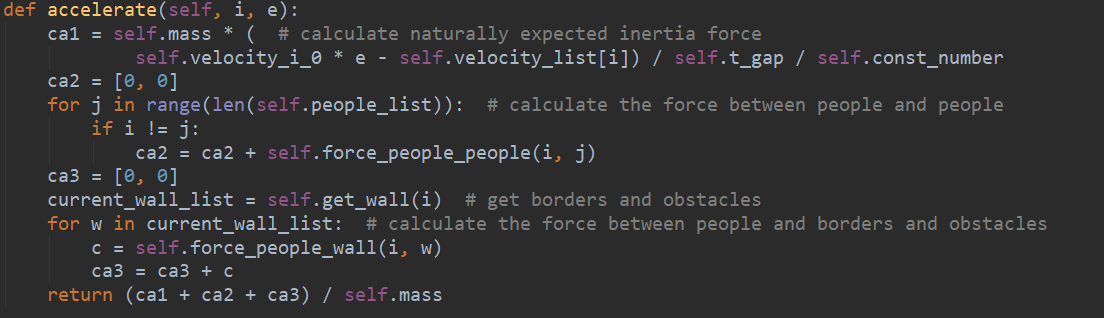
* 注：location是人索引，代表第几个人

获取每个障碍物距离人最近的点

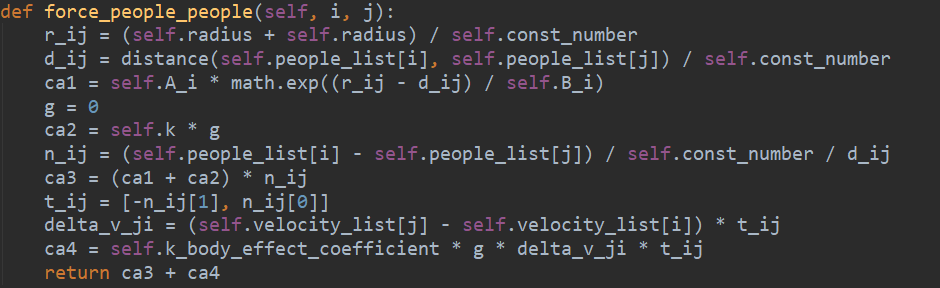


* 注：有多个障碍物，分别计算

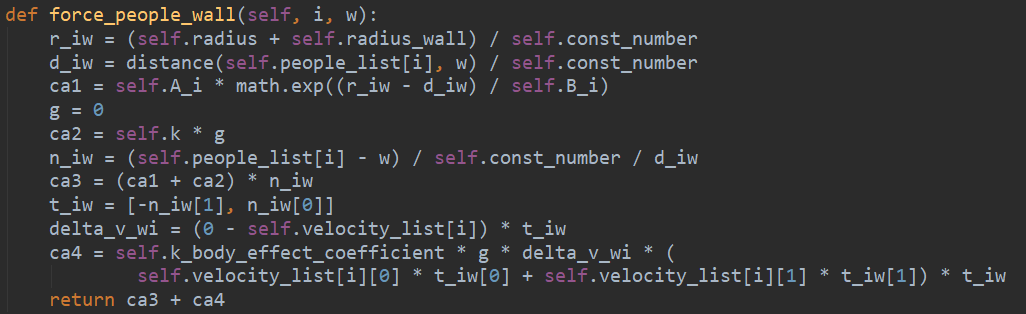
1. 三项力的合理的计算公式



1. 编写行人之间相互作用力的函数



1. 编写行人与墙之间相互作用力的函数



1. 编写搜索算法**（此处放关键代码的截图）**
2. 使用PyQt编写GUI.
3. 根据社会力模型的公式，将以上函数和算法计算出来的结果连接，GUI不断调用社会里计算模块提供的接口函数算出行人在下一时刻所处的位置，在画面上渲染出来。
4. **调试过程**
   1. 调试过程中遇到的第一个问题是在GUI尚未编写完成的情况下该如何验证社会力计算的正确性。我们在实验中采用的方法是，先用单人进行测试，输出每一步社会里计算之后人的位置坐标、速度和加速度，观察有没有超出预计范围的情况。测试通过以后，设置两到三个人进行社会力的测试，通过同样的方法输出并检查每一个人在每一个仿真时刻的信息。
   2. 在一开始的版本中，我们遇到了行人在距离其他行人过近时会被弹开，以及某个行人在两个行人之间反复弹跳的问题。这种情况在行人初始位置过于密集的时候尤为明显。这是由社会力模型的缺陷导致的，如绪言中所述，两个行人之间若有5cm的重叠，将会产生6000N的压力。这么大的力，即使不考虑人的承受能力，也会使人产生巨大的加速度以及随后产生的速度跳变、位置跳变等一系列不符合实际的现象。为了解决这个问题，我们采取了两个措施：一是限制行人初始位置的密集程度，使仿真刚开始时行人之间不会发生重叠；二是限制行人的最大速度及最大加速度，使其不会受到过大的社会力。这两种调整都是符合实际的，在模型仿真时取得了很好的效果，有效消除了原本存在的跳变问题。
   3. 社会力模型中，计算障碍物对人的作用力时，由于障碍物是以一块面积的形式存在的，因此存在多种计算障碍物对人合力的方式。  
      我们最初将障碍物视为一个实心物体，对障碍物内部的每一个网格点依次计算该格点所在位置对行人的作用力，最后将所有网格点的作用力按向量加法进行叠加，算出对行人的合力。然而这种问题存在一定的缺陷，在图形化仿真的过程中，我们发现人在和障碍物还有一段距离时就不再继续向障碍物靠近了，即使人群很密集时依然如此。经过我们的分析，我们认为这是由于如果将障碍物看作一个实心实体，依据原始社会力模型的公式，将会对行人产生一个巨大的合力，因为障碍物中的每一个点都作为了合力的贡献者，即使行人其实并不能感知到障碍物内部的情况。因此我们接下来换了一种方式。  
      在第二次尝试中，我们不再将障碍物看作实心物体，而是看作空心物体，即只有边界会对行人产生作用力。我们对障碍物的四个边界上的所有格点依次按照社会力模型的公式计算其对人的作用力，并将其叠加成合力。这种方法运行结果较第一次尝试好了许多，但仍然存在行人在距墙还有一定距离时便不再前进的问题。究其原因，墙或障碍物（以下简称墙）对行人的力，归根结底只是一种心理力而非物理力，即行人在还没有触碰到墙的时候，就会受到墙对他的心理力。在此过程中，人是无法感知到墙的具体物理属性的，因此墙的实际厚度和行人看不到的部分对心理力并无影响。想明白这个问题之后，我们对计算墙和人之间距离的方式进行了第二次修改。  
      在现在的版本中，我们将墙和障碍物分开处理，对于墙而言，直接计算四个垂直距离，并只按四个垂点计算墙对行人的心理力。对与障碍物而言，我们将人到障碍物之间的距离视为人到障碍物的最近距离，并只按最近距离的那个点计算障碍物对人的心理力。具体的计算方式在上文第三节“数学模型”第四小节“行人与障碍物之间的距离”已经做了具体的阐释，此处不再赘述。通过使用这种方式，我们获得了很好的仿真效果，一个行人在受到其他行人足够大的力时，将有可能贴到墙或障碍物上，很好地模拟了真实情况。
5. **程序运行结果分析（此处需要添加运行结果的截图，对流和疏散各截四张过程图）**