B. Tech Project Report-Stage 1

Simulation of pedestrian dynamics using Cellular Automata

Report by

Rachana Mane (110040009)

Final Year B.tech student

Indian Institute of Technology, Bombay

Under the supervision of

Prof. Gopal R. Patil

23rd November, 2014

Transportation Systems Engineering
Department of Civil Engineering
Indian Institute of Technology Bombay

Acknowledgement

I would really like to express my sincere gratitude towards my project supervisor, **Prof. Gopal R Patil**, for his invaluable suggestions and guidance throughout the project and most importantly for providing me a platform to work with complete freedom.

I would also like to thank all the members of my research group for their help and suggestions.

Date: 24.11.2014

Rachana Mane

110040009

Contents

Abstract

We propose a two-dimensional Cellular Automaton (CA) model to simulate pedestrian dynamics. CA is discrete model used to simulate real life complex system. In this study, pedestrians are considered as an independent entity. Based on the information about the route of the journey and the settings around the pedestrian we can model the movements of the pedestrian under consideration. This process is repeated on for all the pedestrians and hence we can predict the instantaneous movement. Based on the information about the route of the journey and the settings around the pedestrian we can model the movements of the pedestrian under consideration. This process is repeated on for all the pedestrians and hence we can predict the instantaneous movement. In this study, we have considered both space and time to be discrete and quantized to the level of reasonable approximation.

Since movements of pedestrian require more freedom, according to previous study the grid is divided in triangular elements and a pedestrian is assigned to a hexagonal comprising of six triangular elements. This is done to make the simulation close to reality as compared to what is obtained from conventional square elements, where the movements are limited. At this stage the model made for unidirectional flow is studied proposed a for merging and diverging in the second stage.

Chapter 1

Introduction

1.1 Overview

Walking regularly can help you stay healthy and live longer, keep happy, enjoy time with friends and family, learn more about your local area, meet other people, make new friends and look after the environment. And almost everyone can do it, anywhere and at any time, for free. These benefits were even foreseen by the pioneers of the road makers and that is why roman roads had provision for pedestrian walking. Promoting to walk can help in reducing the usage of motorized vehicles, and hence in reducing the carbon emission. Hence walking may be counted as an inevitable factor in sustainable development required by the developing countries. With increasing population and reducing land available, efficient way of right is needed to be provided for safe and comfortable walking that requires study on pedestrian behavior and their flow characteristics. In developing countries, like India, where facilities like shopping mall, airports, parking etc. are being developed, and where population is shooting up provision of efficient facility for pedestrian is unavoidable. Some of the pedestrian facilities were encroached by road side vendors or in un-usable condition making it problematic for pedestrians to make use of the facility.

In this study, we have made an attempt to capture the behavior of pedestrians and develop a mathematical model to simulate pedestrian flow. Presented model is based on Cellular Automaton (CA).

1.2 Study Objective

1. To develop microscopic level rule-set in accordance with the triangular cellular automata for pedestrian flow simulation.

2. To make the simulation model developed by previous student more time and space efficient

3. To develop a logical rule set for bi-directional flows on a straight corridor

4. To introduce realistic situations like sudden stopping, etc. which help in determining Level of Service (LOS) of the facility

5. To retrieve the output data and to utilize it to get various kind of results like pedestrian flow-path, average travel time, average direction change, etc.

6. To calibrate the simulation model developed with more field data

1.3 Motivation

The main motivation of this study is to develop a technique to simulate real time pedestrian behavior on a straight corridor. The simulation results, like average travel time, flow, average deviation from path, etc. could be used both for a quality check and to plan and evaluate future pedestrian facilities efficiently both with respect to time and cost. The motivation for this research is from the day to day life experience at several public places like railway station, malls, etc. where the pedestrian traffic conditions quite poor in our country.

1.4 Organization of thesis

This report is organized in six chapters. Chapter one consist of introduction, objective and background of study .Chapter two describes the literature review. Chapter three describe the methodology developed. Chapter four includes data collection, extraction analysis for study of pedestrian flow characteristics. Chapter five includes summary, conclusion and future scope of work.

Chapter 2

Literature Review

2.1 Different Analogies used

Various analogies have been used to incorporate different behaviors of the pedestrian flows. Henderson [Henderson (1971), (1973) and (1974)] inferred that the behavior of pedestrian crowds is similar to that of gases or fluids. This model adequately takes into account the self organization effects occurring in pedestrian crowds [Helbing et. al (2001)]. An approach using real-coded lattice gas (RLG), which has been developed for fluid simulation, is used for arbitrary velocity and directions for pedestrian dynamics. The lattice gas model is one of main approaches to study pedestrian dynamics and has been extensively studied and applied [Muramatsu et.al (1999)].

2.2 Macroscopic Traffic Flow models

In past simulation was broadly based on the macroscopic models due to its lower complexity. Real-time prediction applications also witnessed the macroscopic model to the above stated benefits. Many researchers attempted to propose models for traffic flow using different analogies. One of them was proposed by Lighthill and Whitham in 1955 where traffic flow was considered to have characteristics similar to compressible fluid, it was commonly known as first-order model [Lighthill et. al (1955), Richards (1956)]. One key issue with the model was that model could simulate by considering traffic density as state variable only, transient behavior was poor. Few modifications were done in the model to overcome these drawbacks like, mean speed was evaluated using dynamic equations [Payne H. J. (1979), Papageorgiou M. (1983), 10]. Lately, simulation has been based on second-order continuum model [Rascle (2002)] and significant amount of work has been done in this area. In these simulations drawbacks of firstorder are addressed, like it can incorporate cars moving backwards, no upper bound on velocity, etc. [Daganzo (1995)]. The continuum model depends on parameters like space mean velocity, flow rate of traffic stream, traffic density and the generation rate.

2.3 Microscopic Models

2.3.1 Introduction

For more than five decades pedestrian crowds have been studied and several attempts have been made to simulate pedestrian flows especially for evacuation purposes using various discrete choice models. Out of all attempts, widely accepted one is cellular due to its discretization of time and space for flow. Set of rules can be set to mimic the reality as close as possible. Another famous model at micro-simulation is social force model that considers the interaction of pedestrians with other pedestrians in realistic manner [Helbing et. al (1995)].

2.3.2 Cellular Automata Model

Cellular Automata simulation is an effective technique for modeling complex emergent collective behavior that is characterized as an Artificial Life Approach to simulation modeling (Adami 1998, Levy 1992). CA is named after the principle of *Automata* (Entities) occupying *cells* according to localized neighborhood rules of occupancy. The CA local rules prescribe the behavior of each automaton creating an approximation of actual individual behavior. Emergent collective behavior is an outgrowth of the interaction the simulation rule set over local neighborhoods. Traditional simulation models apply equation rather than behavioral rules, but CA behavior-based changes of state determine the emergent results. The self-organization in the collective behavior of Artificial Life modeling stems from decentralized sources of decision making such as ant colonies, flocks of birds, etc. CA pedestrian simulation as used here is a parallel distributed, bottom-up approach (Resnick 1994). By designing the CA- based pedestrian from bottom-up at the interface with one another, higher level functions, like route selection and trip behavior, can be added later without fundamentally changing pedestrian dynamics.

The CA interactions of the pedestrians are based on quite understandable behavioral rules. They are easily implemented on computer using any coding language. CA models function as a discrete idealization of partial differential equations that describe fluid flows and allow simulations and interactions that are otherwise intractable (Wolfram, 1994). Only the local rules and sequencing of their use are coded, leaving the many autonomous interactions on the cell matrix to create the emergent macroscopic result. As a result, it has been observed that very few models are capable of capturing essential system features of extraordinary complexity (Bak, 1996).

Over the past few years researchers have demonstrated the applicability of CA micro-simulation to car following and vehicular flows. These CA models have included traffic within a single lane (Nagel and Schreckenberg, 1992), two lane flow with passing (Rickert, et al 1996), bi-direction flow with passing (Simon and Gutowitz, 1998)and network level vehicle flows in the TRANSIMS model (Nagel, Barret and Rickert, 1996). CA traffic models have been shown to provide a good approximation of complex traffic flow over a range of densities including the formation of shock waves in traffic jams.

2.3.3. Social Force Modelling

This theory suggests that in relatively simpler or homogeneous situations, stochastic models can be developed to describe pedestrian's behavior [Weidlich et. al (1983)]. Social Force theory can be used for both continuum and discrete models. The social forces in continuum models can also be visualized as the Newtonian forces where the actual velocity is the resultant of the desired velocity and the interaction forces and personal desire forces [Mehran et. al (2009)]. The concept of fundamental proxemics like a pedestrian's tendency to remain distant from other ones while heading towards his goal is implicitly employed in this theory [Hall (1966)].

2.3.4 Hexagonal Cellular Automata

Recently, some study on the use of hexagonal cellular automata to model evacuation of pedestrians has been done [Yanagisawa et al. (2010)]. HCA has been used to simulate headon collisions of lattice gas particles [Frisch et. al (1986)] within macroscopic limits, wildfire spread [Giuseppe (2004)], debris flows [D'Ambrosio et al. (2003)] characterized by strong inertial effects and many other models. For this project, HCA is preferred over conventional CA as it provides with more movement possibilities and easy lane changing option which is essential for realistic pedestrian flow simulation.

2.4 Cellular Automata rule set

By incorporating a rule set that eliminates anything but critical behavioral factors, the model facilitates the clear understanding of the underlying fundamental dynamics. There are three fundamental elements of pedestrian movements that a bi-directional microscopic model should account for: side stepping (Lane changing), forward movement (braking, acceleration), and conflict mitigation (deadlock avoidance). The basic rule set for the model was developed around these three elements and is designed to work within framework of parallel updates. As used by Rickert et al (1995) and Simon and Gutowitz (1998) lane assignment and forward motions change the position of all pedestrians in two parallel update stages in each time step. Parallel updates avoid succession interdependencies encountered in sequential updates by determining all the new positions before anyone moves. All the entities are then repositioned together. Only the pedestrians in the immediate neighborhood affect the movement of a pedestrian, which though myopic is relatively realistic.

Chapter 3

Model Formation

In this study field data was observed and based on the parameters obtained from the data the proposed model was calibrated. Calibrated model was validated with one site's data and results were acceptable with error within the tolerance level. Proposed model is described in the subsequent sections of this chapter and data collection is articulated in the next chapter.

3.1 Model Formulation

The proposed model is a discrete microscopic model that uses discretized flow space and time to represent the movement of individual pedestrians. There are broadly two factors that affect the movement of a pedestrian viz. static and dynamic. Static factors include goals and obstacles, goals are assumed to attract whereas obstacles to repel. Here a triangular cellular automata model is proposed with a combination of six adjacent triangles (hexagon) as a single pedestrian. The center of this hexagon represents the center of the pedestrian body and the area of the hexagon represents his whole body which if shared by other pedestrian's hexagon would be considered as a collision. Based on static features each cell is assigned a fixed desired for a pedestrian and this value remains fixed. Dynamic features i.e other pedestrians in the vicinity, also influence the movement of pedestrians. Vicinity pedestrians are assumed to create a negative effect resulting in lane change or reduction in speed. Negative effect of vicinity pedestrian depends upon the relative position hence its dynamic in nature. This model has the following advantages over existing models:-

Freedom of speed

Since the number of possible discrete cell has increased due to the triangular grid, the pedestrians have more freedom of speed as the steps are now smaller than the regular HCA (reduced by a factor of √3)

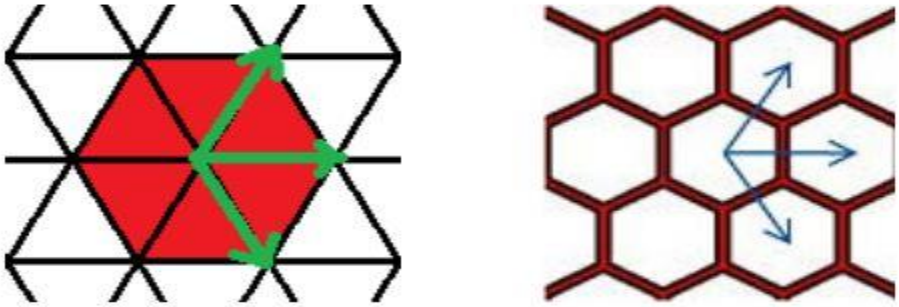


Figure 3.1 Possible forward directions of motion in (a) Triangular CA (b) Hexagonal CA

Closer representation of pedestrian body

Using hexagonal area (by combining six triangular cells) for a pedestrian is a better representation of the space occupied by the pedestrian as compared to conventional square cells [HCM 2000].Since the pedestrian is moving in forward direction, the space required is assumed to be larger than the space required while standing, esp. the length in forward direction. Based on the width dimensions specified, in this study the dimension of triangles have been assumed to be 0.35

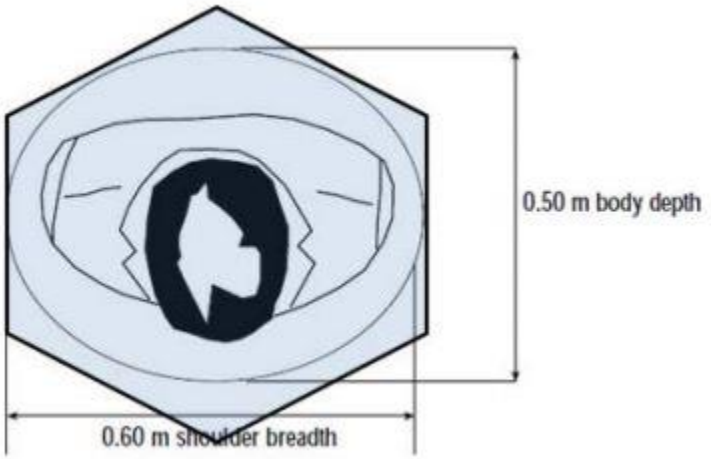


Figure 3.2 Space occupied by the pedestrian for Standing Area, HCM 2000

Partial collision

The triangular grid also results in modeling the collision conditions in a much more realistic way as it allows the possibility of partial overlap for the collision. Real-life collisions are better represented by such partial overlap since a pedestrian does not occupy the other's space completely but only shares only a partial amount of other's space. This is excellent in simulating highly crowded facility where the level of service is very poor and the pedestrians

practically occupy lesser space. The following figure compares this model with hexagonal CA (which is similar in square CA as well)

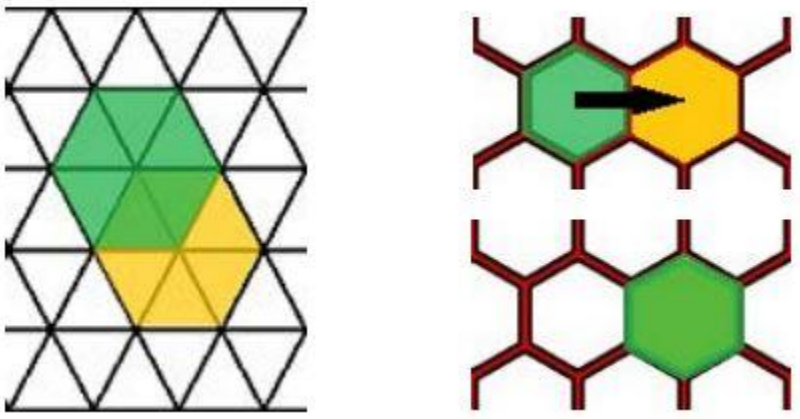


Figure 3.3 Collision type (a) Partial overlap in Triangular CA (b) Complete overlap in Hexagonal CA

Complete coverage of path

Since the most general form of pedestrian facility can be assumed as a rectangular space, the hexagonal cells are unable to complete the whole area while with triangular grid whole space can be used for simulation.

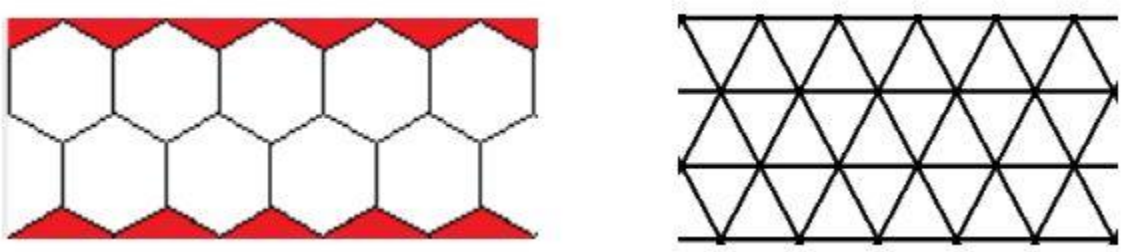


Figure 3.4 the red marked space in left figure cannot be used by pedestrian

The above model has been improvised by incorporating the possibility of lane-changing (lateral movement) by the pedestrians. The lane-changing algorithm is based on the Utility Theory which works on the similar lines of discrete floor field method [Burstedde et al. (2001)]. According to utility theory, the pedestrian has a certain utility or desirability assigned to every cell where he can move in the next time-step. In this model, it is assumed that the pedestrian does not always choose the cell with maximum desirability but selects between the three most desirable cells based on the logit model. This is because the pedestrian cannot precisely estimate the best choice for him at every step but can only guesstimate among his best few options.

Freedom of movement

In a straight corridor, a pedestrian is inclined to follow a straight path unless he encounters a movable or immovable obstacle. In such cases, he tends to deviate from his path to avoid any collision. In this study, this deviation is within 120 (-60 to +60)degrees to a certain set of points based on the maximum velocity of the pedestrian. The following figure illustrates the set of points (marked in black) and the area (yellow) where a pedestrian (red) with maximum velocity of 1.20 m/s (4 units per second, i.e. 4x0.30m) can travel:

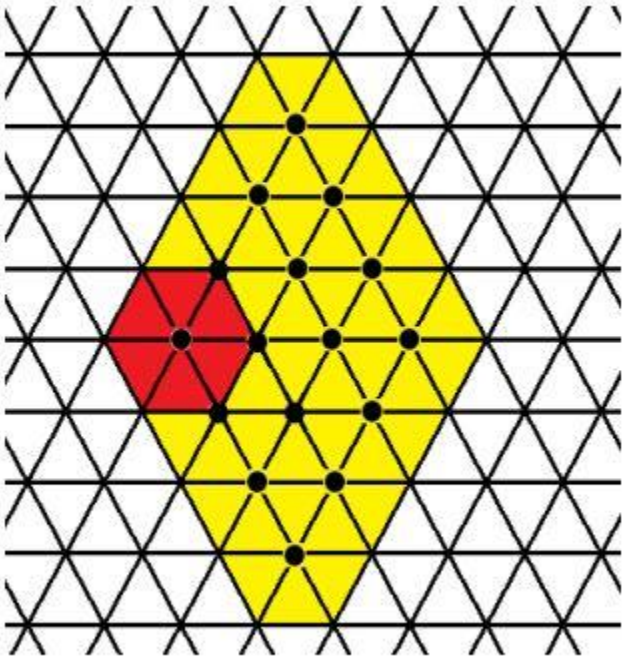


Figure 3.5 Set of points where a pedestrian with maximum v = 1.20 m/s can move in 1 time-step

3.1.1 Cell arrangement and Coordinate system

The arrangement of cells and the possible pedestrian movements considered for the present study is shown in the figure below:

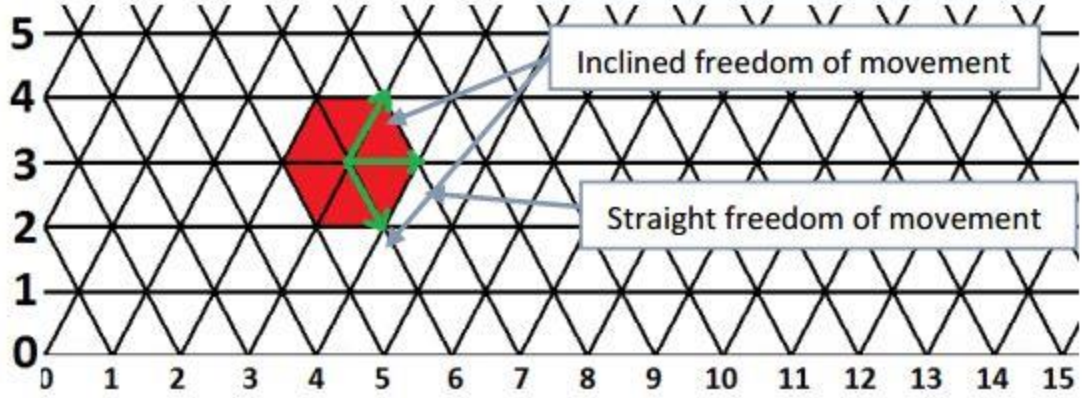


Figure 3.6 Grid showing absolute X and virtual Y coordinates

3.2 Bi-directional Rule set:

3.2.1 Lane change (parallel update 1)

1) Eliminate Conflicts: Empty cell in between two cells is available to only one (50/50 random assignment)

2) Identify gaps: Lane is chosen such that it best advances forward movement upto v_{max}

a) For Dynamic multiple lanes (DML):

(i) Step out of lane of a walker from opposite direction by assigning gap=0 if within 8 cells.

(ii) Step behind a same direction walker by choosing any available lane with gap_same = 1 when gap = 1.

b) ties of equal maximum gaps ahead are solved according to:

(i) Two-way tie between the adjacent lanes: 50/50 random assignment.

(ii) Two-way tie between current lane and single adjacent lane: stay in lane

(iii)Three-way tie: Stay in lane

3) Move: Each pedestrian P_n is moved +1, 0, -1 lateral sidestep after 1-3 is complete.

3.2.2 Step forward (Parallel update 2)

1) Update velocity: Let $v(P_n)$ = gap where gap is from gap computation section below.

2) Exchanges: IF gap=0 or 1 AND gap = gap_opp(Cell occupied by an opposite direction pedestrian) THEN with probability p_exchg $v(P_n)$ = gap + 1 ELSE $v(P_n) = 0$.

3) Move: Each pedestrian P_n is moved $v(P_n)$ cells forward on the lattice.

3.3.3 Gap Computation

1) Same direction: Look ahead a max of 8 cells(8=2*largest v_max) IF occupied cell found with same direction THEN set gap_same to number cells between entities ELSE gap_same = 8

2) Opposite direction: IF occupied cell found with same direction THEN set gap_opp to INT (0.5*number of cells between entities) ELSE gap_opp = 4

3) Assign gap = MIN (gap_same, gap_opp, v_max)

Chapter 4

Data Collection

4.1 General

In our daily life, we start and the trip with pedestrians, the study of Pedestrian characteristics is very important. Every pedestrian react differently to the different conditions like hindrances, talking or texting on phone, listening to music etc. There particular reactions depend of the gender, age and purpose of the trip.

Video-graphic technique was employed for collecting the pedestrian data. Recordings were observed at a stretch of 7m distance. Video camera was kept at an elevated level to capture the entire stretch and to make head count simpler. The movements of pedestrians were recorded during the afternoon time of 2.00 P.M in Hiranandani area near Galleria mall on a working day. Pedestrians were counted and their walking speeds and relevant characteristics were recorded at different walking facilities. Looking at the continuous flow of pedestrians entering the trap the flow data was extracted on five minute basis. The time taken by each pedestrian to cross the trap length was noted to an accuracy of 0.01s to determine pedestrian speed.

4.2 Details of the study location

Site (Location)	**Width of Pedestrian facility**	**Physical condition**	**Weather**
Hiranandani (Galleria Mall)	3 meters	Rough, Levelled, well maintained	Hot, sunny

Figure 4.2 Footpath picture at Hiranandani (Galleria Mall sidewalk)

4.3 Analysis of pedestrian characteristic

Around 310 pedestrians on this site were observed for speed calculations. Pedestrians were observed for every 5 minutes interval. Pedestrians observed were engaged in some activity:

texting, talking, listening to music, talking to each other, smoking, drinking etc. College groups or in general groups were seen to move slowly as compared to individual pedestrian. Females were seen to move comparatively slowly and in groups. Couples who were talking or holdings hands were comparatively slower.

4.4 Number of pedestrians and their mean speed

Site(Location)	Female		Male	
	Count	Avg. speed(km/hr)	Count	Avg. speed(km/hr)
	56	1.57	254	2.31

4.5 Calibration

Data collection at this location was used to calibrate the model. Since in real world the space taken by a pedestrian doesn't depend on the pedestrian himself, effect of density, flow rate are also need to be considered. Due to these reason we can be flexible with size of the cellular automata. The cell size in this study is considered to be dependent on the actual pedestrian flow. Other parameters like, factor of reduction in desirability due to lane change were also manipulated. Following table gives the comparison of the field data to the data obtained from field.

Chapter 5

Results and Discussion

5.1 Results

The output file contains position in space of all the pedestrians at every time-step, hence it is possible to trace back the path of any of the pedestrian, compute the number of collisions or close calls, compute average travel time and thus average delay per pedestrian, calculate the average space available to every pedestrian and hence the LOS of the facility. In commercial area effect of age and baggage on speed is less significant.

Mathematical formulation for bi-directional pedestrian flow is formulated.

Chapter 6
Future work

Work done up to this stage comprises of establishing a basic formulation of a mathematical model to simulate bi-directional pedestrian flow on a corridor. This model was further calibrated with data extracted from field data from Hiranandani (Galleria). In order to make model robust and more generic in nature, modification to the formulation is required through field data encompassing the wide spectrum of flow rate under different environmental conditions. Another aspect that needs to be added to the model includes merging, diverging section. Apart from the mathematical formulation, user friendly graphic interface is to be developed. More field observations to be made to calibrate with model.

The mathematical formulation is to be carried out to develop a code in Java for simulation in stage 2.

References

B. Victor and A. Jeffrey, “Cellular Automata Micro-simulation of Bi-Directional Pedestrian Flows”, NYSDT, NY, USA

B. D. Hankin, R. A.Wright,1958, “Passenger flowing subways'' Operational Research Quarterly 9 81 - 88Hoel, 1968

Burstedde et al., “Simulation of pedestrian dynamics using a two-dimensional cellular automaton”, Physica A 295 (2001) 507–525

A.T. Giuseppe (2004), “Predicting Wildfire Spreading Through a Hexagonal Cellular Automata Model” ACRI 2004, LNCS 3305, pp. 385–394

A Polus, J L Schofer, A Ushpiz,1983, Pedestrian flow and level of service'' Journal of Transportation Engineering 109 46 – 56

C. Daganzo (1995), “Requiem for second-order fluid approximation to traffic flow”, Transp. Res. B 29B (4), 277-286

Chapter 11 and Chapter 18, Highway Capacity Manual 2000 ,TRB

D Boeminghaus, 1982 FuÞgangerbereiche + Gestaltungselemente/Pedestrian Areas and Design Elements/Zones pour Pie¨tons + Eè lements de Conception (Kramer, Stuttgart)

Dynamics Springer-Verlag, Berlin. pp. 173-181

Dijkstra et al. (2001), "A Multi-Agent Cellular Automata Model of Pedestrian Movement.", In M. Schreckenberg and S.D. Sharma(ed.): Pedestrian and Evacuation

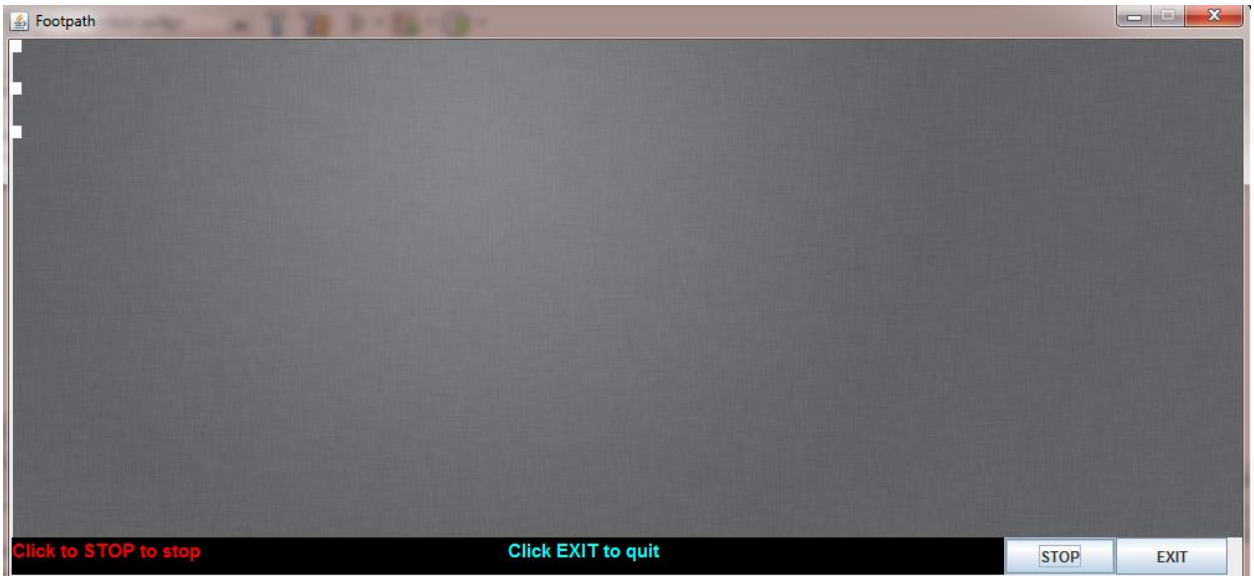
D’Ambrosio et al., “Simulating debris flows through a hexagonal cellular automata model”, Natural Hazards and Earth System Sciences (2003) 3: 545–559

D Helbing, P Molna¨r, I J Farkas, K Bolay, Self-organizing pedestrian movement, Environment and Planning B: Planning and Design 2001, volume 28, pages 361 – 383

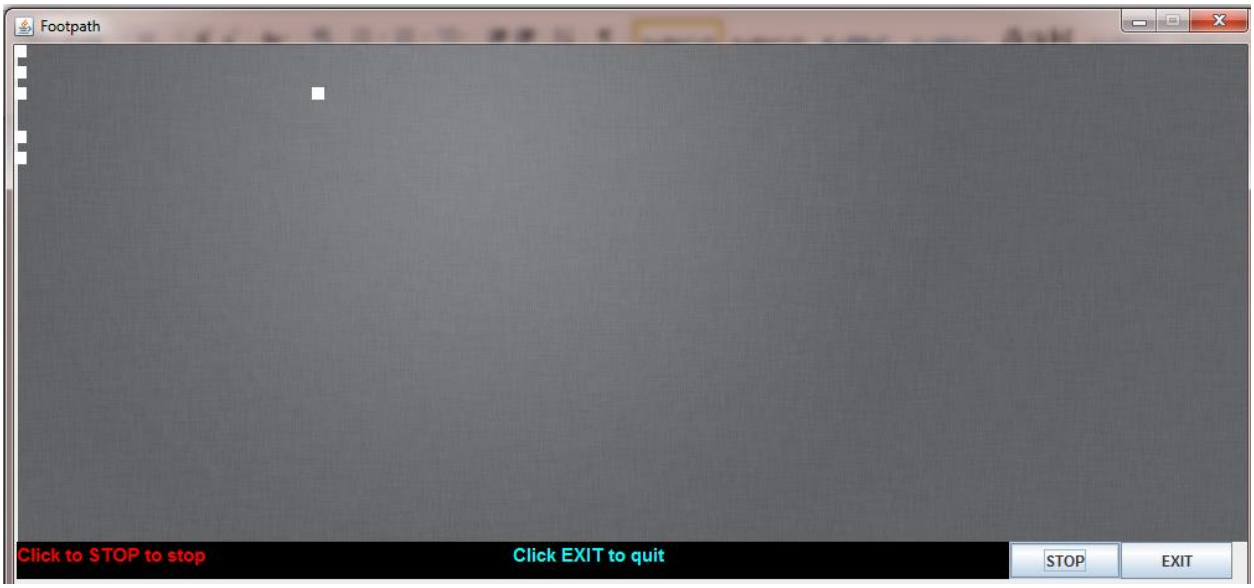
Appendix A

Screenshots of model

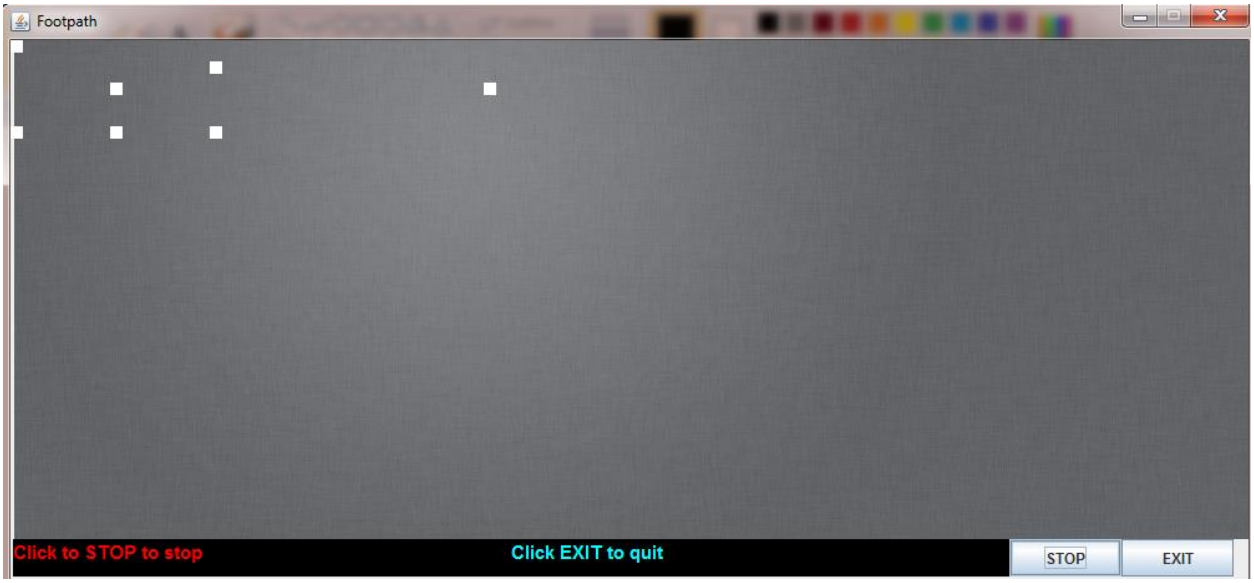
1.



2.



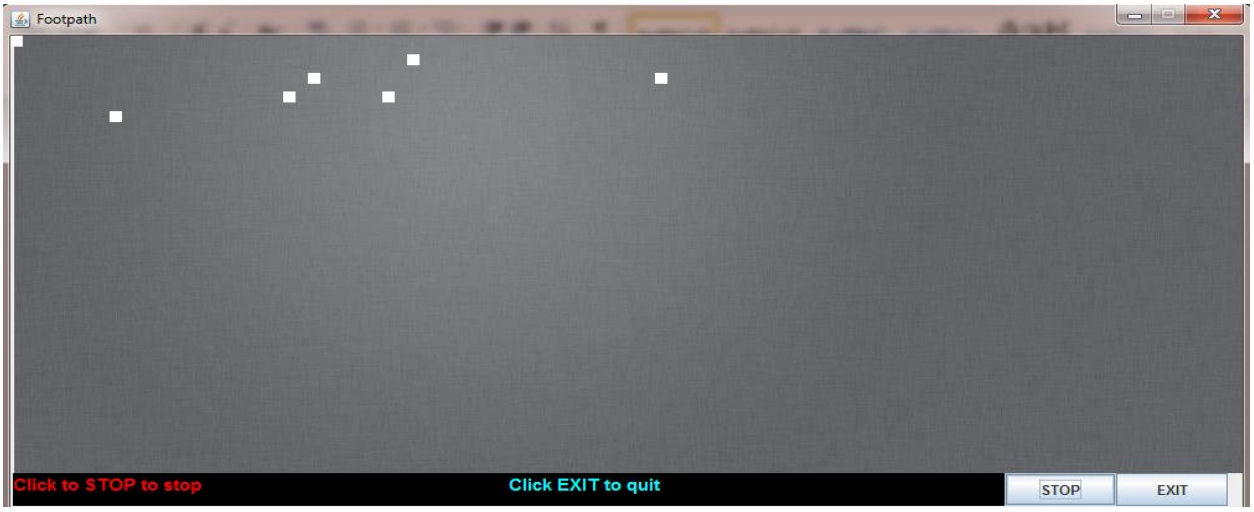
3.


Footpath
Click to STOP to stop
Click EXIT to quit
STOP
EXIT

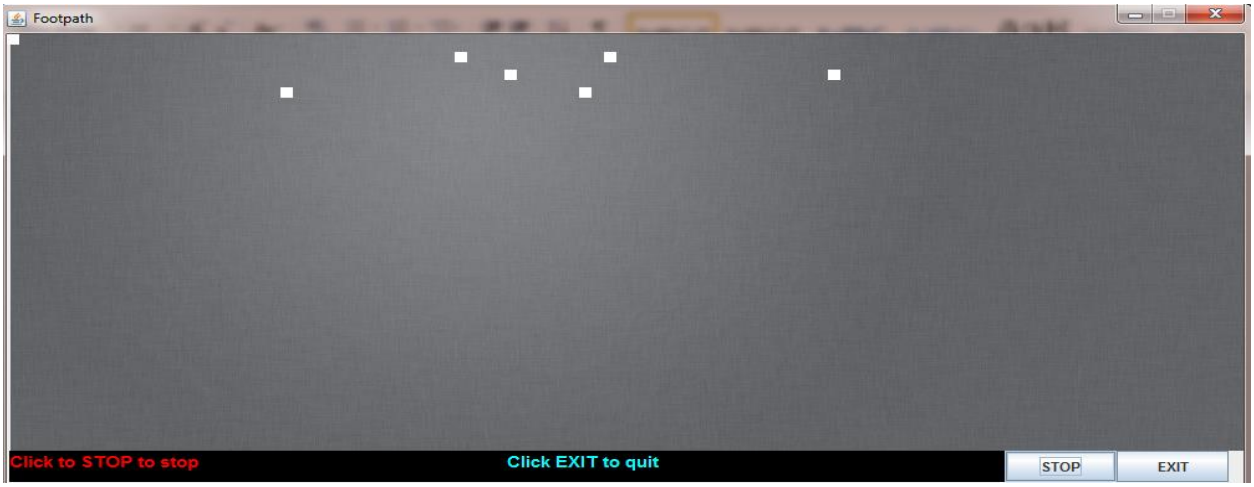
4.


Footpath
Click to STOP to stop
Click EXIT to quit
STOP
EXIT

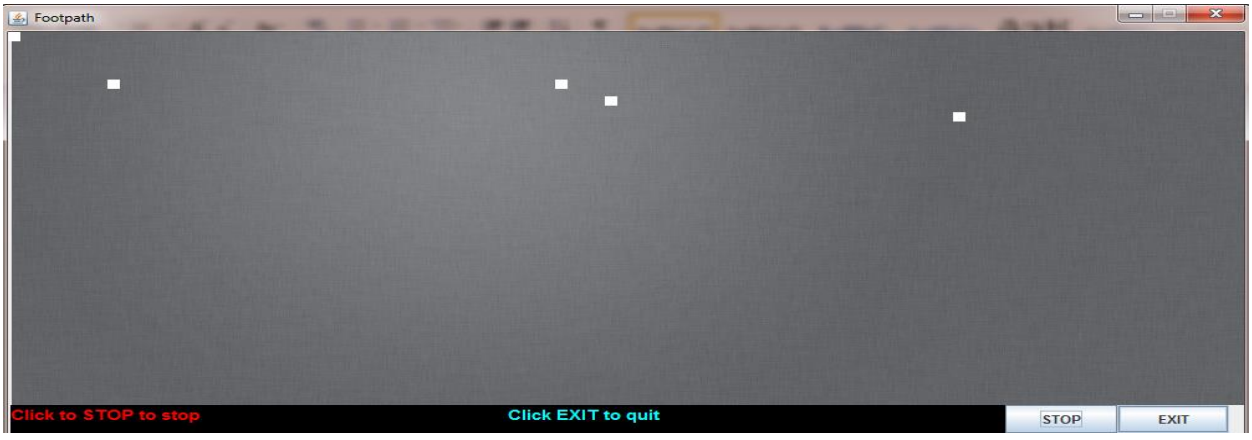
5.


Footpath
Click to STOP to stop
Click EXIT to quit
STOP
EXIT

6.


Footpath
Click to STOP to stop
Click EXIT to quit
STOP
EXIT

7.


Footpath
Click to STOP to stop
Click EXIT to quit
STOP
EXIT

Appendix B

1. Java code for Lane Change model

```
package pedestrian_flow;

import java.io.*;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

// ----------------------------------------------------------------------------------------------------------------
-

public class Lane_change

{

   static final double width = 2.5; // meter width of the corridor under study

   static final double ped_flow =800;  // pedestrian/hour (pedestrian flow)

//----------------------------------------------------------------------------------------------------------------
----------

   static double Tsize = 0.37-(4*ped_flow/100000);//length of the sides of triangular cell as a
function of the pedestrian flow rate

   static double vmax = 4;  // maximum a pedestrian can move is vmax cells in one step

   static final int time_step = 1; // variable to specify the increment in the time step

   static int steps=0; // initiating the time of study

   static int total_steps= 10*60; //total simulation time

   static double spot = 45; // point of analysis

   static int pedestrian_count=0; // initiating the counter for total pedestrians

   static int timeNext = 1; // variable for time keeping during simulation
```

```
   static double bas_lc =-1.5;// weight to the lane changing effect in basic desire of a
pedestrian

   static double bas_l =1.5;// weight to the straight movement in basic desire of a
pedestrian

   static double inter_lc =3;// weight to the lane changing effect in interference matrix of a
pedestrian

   static double inter_l=2;// weight to the straight movement effect in interference matrix of
a pedestrian

   static int interference_count = -1;  // 4 * vmax^2 + 12*vmax + 6

   static int base_des_count = -1;  // vmax^2 + 2*vmax

//-------------------------------------------------------------------------------------------------------------------
----------

// function is self explanatory

public static double power(int a, int b){

   return Math.pow(ped_flow*time_step/3600, a) * Math.exp(-
ped_flow*time_step/3600)/b;

   }

     //-------------------------------------------------------------------------------------------------------------------
--------------

public static double [][][] bdesmat(double vmax, double [][][] basic_desirability_matrix, int
ped_count)

      throws FileNotFoundException{


      try ( // Function to initiate the matrix with basic desire for each pedestrian

            PrintWriter basedesmat = new PrintWriter("base_mat.txt") // Writing basic desire
matrix in a file named "base_mat.txt"

      ) {
```

```
        basedesmat.println("This is base des mat");
        for (int i = 1; i <= ped_count; i++) {
           int temp = -1;

           for (int d =0 ; d <= vmax; d++){
              for (int v = d;v <= vmax; v++){
                 temp++;
                 basic_desirability_matrix[i][temp][0] = v-d*0.5;
                 basic_desirability_matrix[i][temp][1] = d;  //*ydir; // ydir=1
                 basic_desirability_matrix[i][temp][2] =  12 + v*bas_l + d*bas_lc;
                 basedesmat.println(basic_desirability_matrix[i][temp][0]+"\t"+
basic_desirability_matrix[i][temp][1] +"\t"+ basic_desirability_matrix[i][temp][2]);
              }
           }
           for (int d = 1; d <= vmax; d++){
              for (int v=d;v <= vmax; v++){
                 temp++;
                 basic_desirability_matrix[i][temp][0] = v-d*0.5;
                 basic_desirability_matrix[i][temp][1] = -d;  //*ydir; // ydir=-1
                 basic_desirability_matrix[i][temp][2] =  12 + v*bas_l + d*bas_lc;
                 basedesmat.println(basic_desirability_matrix[i][temp][0]+"\t"+
basic_desirability_matrix[i][temp][1] +"\t"+ basic_desirability_matrix[i][temp][2]);
              }
           }
           base_des_count = temp;
```

```
        }
      }
    return basic_desirability_matrix;
}
//-----------------------------------------------------------------------------------------------------------
public static double [][][] intmat(double vmax, double [][][] interference_matrix, int ped_count)
      throws FileNotFoundException{
    // Function to initiate the matrix with negative effect by each pedestrian on a nearby pedestrian
    try (PrintWriter basedesmat = new PrintWriter("base_mat_int.txt") // Writing interference matrix in a file named "base_mat.txt"
    ) {
       // Function to initiate the matrix with negative effect by each pedestrian on a nearby pedestrian
       basedesmat.println("This is base int mat");
       for (int j = 1; j <= ped_count; j++) {
          int i = -1;
          for (double v =-(vmax+1); v<=(vmax+1);v++){
             i++;
             interference_matrix[j][i][0] = v;
             interference_matrix[j][i][1] = 0;
             interference_matrix[j][i][2] = inter_l*(Math.abs(v))-12;
             basedesmat.println(interference_matrix[j][i][0]+"\t"+ interference_matrix[j][i][1]+"\t"+ interference_matrix[j][i][2]);
```

```
            }
            // for lane change = 0
            for (double d = 1; d<=(2*vmax+1);d++){
               for (double v=-(vmax+1-d/2);v<=(vmax+1-d/2);v++){
                  i++;
                  interference_matrix[j][i][0] = v;
                  interference_matrix[j][i][1] = d; //(*ydir;) // int ydir = 1;
                  interference_matrix[j][i][2] =  d*inter_lc+inter_l*(Math.abs(v))-20;
                  basedesmat.println(interference_matrix[j][i][0]+"\t"+
interference_matrix[j][i][1] +"\t"+ interference_matrix[j][i][2]);
               }
            }
            // for lane change = 1
            for (double d = 1; d<=(2*vmax+1);d++){
               for (double v=-(vmax+1-d/2);v<=(vmax+1-d/2);v++){
                  i++;
                  interference_matrix[j][i][0] = v;
                  interference_matrix[j][i][1] = -d; //(*-ydir;) // int ydir = -1;
                  interference_matrix[j][i][2] = d*inter_lc+inter_l*(Math.abs(v))-20;
                  basedesmat.println(interference_matrix[j][i][0]+"\t"+
interference_matrix[j][i][1] +"\t"+ interference_matrix[j][i][2]);
               }
            }
            interference_count = i;
         }
```

```
        }
      return interference_matrix;
}
      //-----------------------------------------------------------------------------------------------------------
---------------
public static double [][][][] negeff(double [][][][] returnarray,int pedestrian_count,int
total_steps,
        double [][][] basic_desirability_matrix, double [][][] interference_matrix,double [][][]
main_matrix,int time,double vmax, double ywidth)
        throws FileNotFoundException{
// This function checks for the pedestrian in the vicinity of pedestrian under consideration.
//This function performs its function for each pedestrian on the floor
   double [][][][] vicinityPed = new double [(2+total_steps)][(1+pedestrian_count)][99][5];
   int i = 1;
   int flag1 = 1;
   if(main_matrix[time][i][0] == 0){
      flag1 = 0;
   }
   while(flag1 == 1){
      int vicinity_pedestrian_count=0;
      int j = 1;
     int flag2 = 1;
      if(main_matrix[time][j][0] == 0){
         flag2 = 0;
      }
```

```
while(flag2 == 1){

  if (j != i){

      double relativeX = main_matrix[time][i][1] - main_matrix[time][j][1];

      double relativeY = main_matrix[time][i][2] - main_matrix[time][j][2];

      if(Math.abs(relativeX) + Math.abs(relativeY) / 2 <= 4 && Math.abs(relativeY) <= 7 &&
main_matrix[time][j][2]*main_matrix[time][j][2]+main_matrix[time][j][1]*main_matrix[tim
e][j][1] != 0 && main_matrix[time][i][1]<main_matrix[time][j][1] ){

          vicinity_pedestrian_count++;

          for (int n = 0; n <= 4; n++){

          vicinityPed[time][i][vicinity_pedestrian_count][n] = main_matrix[time][j][n];


          }

        double[][][] vicinity_interference_matrix = new double
[999][interference_matrix[j].length][3];

        for (int k = 0; k < interference_count; k++){

          for (int n = 0; n <= 2; n++){


  vicinity_interference_matrix[vicinity_pedestrian_count][k][n]=interference_matrix[j][k][
n];

          }

        }

        for (int k = 0; k <interference_count; k++){

          vicinity_interference_matrix[vicinity_pedestrian_count][k][0] += relativeX;
  // shifting the X coordinates of Interference matrix as per the Pedestrian

          vicinity_interference_matrix[vicinity_pedestrian_count][k][1] += relativeY;
  // shifting the Y coordinates of Interference matrix as per the Pedestrian

          for (int l = 0; l < base_des_count; l++){
```

```
                    if (vicinity_interference_matrix[vicinity_pedestrian_count][k][0] ==
basic_desirability_matrix [i][l][0] &&
vicinity_interference_matrix[vicinity_pedestrian_count][k][1] == basic_desirability_matrix
[i][l][1]){

                    basic_desirability_matrix [i][l][2] = basic_desirability_matrix [i][l][2] +
vicinity_interference_matrix[vicinity_pedestrian_count][k][2];

                    }

            }

            }

        }

    }

     j++;

     if(main_matrix[time][i][0] == 0||j>pedestrian_count){

        flag2 = 0;

    }

  }

  for (int k = 0; k < base_des_count; k++){


    if ((basic_desirability_matrix [i][k][1] + main_matrix[time][i][2]) > (ywidth - 1) ||
(basic_desirability_matrix [i][k][1] + main_matrix[time][i][2]) < 1){

        basic_desirability_matrix [i][k][2] = -1000;


     }

  }

  i++;

  if(main_matrix[time][i][0] == 0||i>pedestrian_count){

    flag1 = 0;
```

```
    }
}
    returnarray[0]=basic_desirability_matrix;
    returnarray[1]=main_matrix;
    return returnarray;
    // Final output of the function is the updated basic desirability matrix.
}
//-------------------------------------------------------------------------------------------------------------------
public static double [][][] updatePosition(double ywidth,int pedestrian_count, double [][][] main_matrix,
        int timeNext, int time, double [][][] basic_desirability_matrix) throws FileNotFoundException{
    // Based of the updated basic desire matrix, this function invokes logit model and accordingly update the position of each pedestrian
    PrintWriter averagespeed= new PrintWriter("Average_speed.txt");  // saving pedestrian data: ped_count, X, Y, random, V
    averagespeed.println( "Ped ID" + "\t"+ "Velocity");
    int i=1;
    int flag1 = 1;
    if(main_matrix[time][i][0] == 0){
        flag1 = 0;
    }
    while(flag1 == 1){
        ArrayList<ArrayList<Double>> sortedList = new ArrayList<ArrayList<Double>>();
    for (int j = 0; j <=base_des_count ; j++){
```

```
    ArrayList<Double> temp = new ArrayList<Double>();
    temp.add(basic_desirability_matrix[i][j][0]);
    temp.add(basic_desirability_matrix[i][j][1]);
    temp.add(basic_desirability_matrix[i][j][2]);
    sortedList.add(temp);
  }
     Collections.shuffle(sortedList);
     Collections.sort(sortedList, new Comparator<ArrayList<Double>>(){
        public int compare(ArrayList<Double> a, ArrayList<Double> b){
           return -Double.compare(a.get(2), b.get(2));
        }
     }
     );
    double probabSum =
Math.exp(sortedList.get(0).get(2))+Math.exp(sortedList.get(1).get(2))+Math.exp(sortedList.
get(2).get(2));// LOGIT
    //double probabSum = sorted.get(0).get(2)+sorted.get(1).get(2)+sorted.get(2).get(2);

    if (sortedList.get(0).get(2)*sortedList.get(1).get(2)*sortedList.get(2).get(2)>-1000000){
        double random =Math.random();
        if (random <= (  Math.exp(sortedList.get(0).get(2)) / probabSum)){
           // Probability as per LOGIT
              int q=0;
              if(main_matrix[time][i][2] + sortedList.get(q).get(1)>= 0 &&
main_matrix[time][i][2] + sortedList.get(q).get(1)<6){
```

```
if ( main_matrix[time][i][1] < spot &&  main_matrix[time][i][1] +
sortedList.get(q).get(0) > spot ){

averagespeed.println(main_matrix[timeNext][i][0] +
"\t"+sortedList.get(q).get(0)+Math.abs((sortedList.get(q).get(1))/2));

}

main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(q).get(0);

main_matrix[timeNext][i][2] = main_matrix[time][i][2] + sortedList.get(q).get(1);

main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((sortedList.get(q).get(1))/2);

}

else if(main_matrix[time][i][2] + sortedList.get(q).get(1)< 0){

if ( main_matrix[time][i][1] < spot &&  main_matrix[time][i][1] +
sortedList.get(q).get(0) > spot ){

averagespeed.println(main_matrix[timeNext][i][0] +
"\t"+sortedList.get(q).get(0)+Math.abs((sortedList.get(q).get(1))/2));

}

main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(q).get(0);

main_matrix[timeNext][i][2] =0;

main_matrix[timeNext][i][4] = Math.abs(sortedList.get(q).get(0))+Math.abs((0-
main_matrix[timeNext][i][2])/2);

}

else if (main_matrix[time][i][2] + sortedList.get(q).get(1)>5){

if ( main_matrix[time][i][1] < spot &&  main_matrix[time][i][1] +
sortedList.get(q).get(0) > spot ){

averagespeed.println(main_matrix[timeNext][i][0] +
"\t"+sortedList.get(q).get(0)+Math.abs((sortedList.get(q).get(1))/2));

}
```

```
            main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(q).get(0);

            main_matrix[timeNext][i][4] = Math.abs(sortedList.get(q).get(0))+Math.abs((5-
main_matrix[timeNext][i][2])/2);

            main_matrix[timeNext][i][2] = 5;

        }

            for (int j = 1; j < i; j++){

            if (main_matrix[timeNext][j][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j][2] == main_matrix[timeNext][i][2] ){

                    q=1;

                    if(main_matrix[time][i][2] + sortedList.get(q).get(1)>= 0 &&
main_matrix[time][i][2] + sortedList.get(q).get(1)<6){

                main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                main_matrix[timeNext][i][2] = main_matrix[time][i][2] +
sortedList.get(q).get(1);

                main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs(((sortedList.get(q).get(1))/2));

            }

                    else if(main_matrix[time][i][2] + sortedList.get(q).get(1)< 0){

                main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                main_matrix[timeNext][i][2] =0;

                main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs(((0-main_matrix[timeNext][i][2])/2));

                    }

                    else if (main_matrix[time][i][2] + sortedList.get(q).get(1)>5){
```

```
                main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                        main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((5-main_matrix[timeNext][i][2])/2);

                        main_matrix[timeNext][i][2] = 5;

                    }

                    for (int j1 = 1; j1 < i; j1++){

                if (main_matrix[timeNext][j1][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j1][2] == main_matrix[timeNext][i][2] ){

                        q=2;

                        if(main_matrix[time][i][2] + sortedList.get(q).get(1)>= 0 &&
main_matrix[time][i][2] + sortedList.get(q).get(1)<6){

                    main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                    main_matrix[timeNext][i][2] = main_matrix[time][i][2] +
sortedList.get(q).get(1);

                    main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((sortedList.get(q).get(1))/2);

                        }

                        else if(main_matrix[time][i][2] + sortedList.get(q).get(1)< 0){



                    main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                    main_matrix[timeNext][i][2] =0;

                    main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs(((0-main_matrix[timeNext][i][2])/2));

                        }
```

```
                    else if (main_matrix[time][i][2] + sortedList.get(q).get(1)>5){

                main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((5-main_matrix[timeNext][i][2])/2);

                main_matrix[timeNext][i][2] = 5;

                    }

                    for (int j11 = 1; j11 < i; j11++){

                if (main_matrix[timeNext][j11][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j11][2] == main_matrix[timeNext][i][2] ){

                   main_matrix[timeNext][i][1] = main_matrix[time][i][1];

                            main_matrix[timeNext][i][2] = main_matrix[time][i][2];

                   main_matrix[timeNext][i][4] = 0;

                }

                    }

            }

               }

        }

        }

    }

    else if (random <= ((Math.exp(sortedList.get(0).get(2)) +
Math.exp(sortedList.get(1).get(2))) / probabSum)){

        int q=1;

        if(main_matrix[time][i][2] + sortedList.get(q).get(1)>= 0 &&
main_matrix[time][i][2] + sortedList.get(q).get(1)<6){

        main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(q).get(0);
```

```
main_matrix[timeNext][i][2] = main_matrix[time][i][2] + sortedList.get(q).get(1);

main_matrix[timeNext][i][4] =Math.abs(
sortedList.get(q).get(0))+Math.abs(((sortedList.get(q).get(1))/2));

}

else if(main_matrix[time][i][2] + sortedList.get(q).get(1)< 0){

main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(q).get(0);

main_matrix[timeNext][i][2] =0;

main_matrix[timeNext][i][4] = Math.abs(sortedList.get(q).get(0))+Math.abs((0-
main_matrix[timeNext][i][2])/2);

}

else if (main_matrix[time][i][2] + sortedList.get(q).get(1)>5){

main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(q).get(0);

main_matrix[timeNext][i][4] = Math.abs(sortedList.get(q).get(0))+Math.abs((5-
main_matrix[timeNext][i][2])/2);

main_matrix[timeNext][i][2] = 5;

}

for (int j = 1; j < i; j++){

if (main_matrix[timeNext][j][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j][2] == main_matrix[timeNext][i][2] ){

   q=2;

        if(main_matrix[time][i][2] + sortedList.get(q).get(1)>= 0 &&
main_matrix[time][i][2] + sortedList.get(q).get(1)<6){

    main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

    main_matrix[timeNext][i][2] = main_matrix[time][i][2] +
sortedList.get(q).get(1);
```

```
                main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((sortedList.get(q).get(1))/2);

                    }

                    else if(main_matrix[time][i][2] + sortedList.get(q).get(1)< 0){

                main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                main_matrix[timeNext][i][2] =0;

                main_matrix[timeNext][i][4] = sortedList.get(q).get(0)+Math.abs((0-
main_matrix[timeNext][i][2])/2);

                    }

                    else if (main_matrix[time][i][2] + sortedList.get(q).get(1)>5){

                main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((5-main_matrix[timeNext][i][2])/2);

                main_matrix[timeNext][i][2] = 5;

                    }

                    for (int j1 = 1; j1 < i; j1++){

                if (main_matrix[timeNext][j1][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j1][2] == main_matrix[timeNext][i][2] ){

                            q=0;

                            if(main_matrix[time][i][2] + sortedList.get(q).get(1)>= 0 &&
main_matrix[time][i][2] + sortedList.get(q).get(1)<6){

                      main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                      main_matrix[timeNext][i][2] = main_matrix[time][i][2] +
sortedList.get(q).get(1);
```

```
                    main_matrix[timeNext][i][4] =  Math.abs(sortedList.get(q).get(0))+
Math.abs((sortedList.get(q).get(1))/2);

                        }

                        else if(main_matrix[time][i][2] + sortedList.get(q).get(1)< 0){

                    main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                    main_matrix[timeNext][i][2] =0;

                    main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(q).get(0))+Math.abs((0-main_matrix[timeNext][i][2])/2);

                }

                        else if (main_matrix[time][i][2] + sortedList.get(q).get(1)>5){

                    main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(q).get(0);

                    main_matrix[timeNext][i][4] =  Math.abs(sortedList.get(q).get(0))+
Math.abs((5-main_matrix[timeNext][i][2])/2);

                    main_matrix[timeNext][i][2] = 5;

                }

                        for (int j11 = 1; j11 < i; j11++){

                    if (main_matrix[timeNext][j11][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j11][2] == main_matrix[timeNext][i][2] ){

                                main_matrix[timeNext][i][1] = main_matrix[time][i][1];

                                main_matrix[timeNext][i][2] = main_matrix[time][i][2];

                                main_matrix[timeNext][i][4] = 0;

                    }

                        }

            }
```

```
                    }
                }
            }
        }
        else{
            main_matrix[timeNext][i][1] = main_matrix[time][i][1] + sortedList.get(2).get(0);
            main_matrix[timeNext][i][2] = main_matrix[time][i][2] + sortedList.get(2).get(1);
            main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(2).get(0))+Math.abs((sortedList.get(2).get(1))/2);

            for (int j = 1; j < i; j++){
            if (main_matrix[timeNext][j][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j][2] == main_matrix[timeNext][i][2]){
                    main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(1).get(0);
                    main_matrix[timeNext][i][2] = main_matrix[time][i][2] +
sortedList.get(1).get(1);
                    main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(1).get(0))+Math.abs((sortedList.get(1).get(1))/2);
                    for (int j1 = 1; j1 < i; j1++){
                if (main_matrix[timeNext][j1][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j1][2] == main_matrix[timeNext][i][2] ){
                        main_matrix[timeNext][i][1] = main_matrix[time][i][1] +
sortedList.get(0).get(0);
                        main_matrix[timeNext][i][2] = main_matrix[time][i][2] +
sortedList.get(0).get(1);
                        main_matrix[timeNext][i][4] =
Math.abs(sortedList.get(0).get(0))+Math.abs((sortedList.get(0).get(1))/2);
```

```
                        for (int j11 = 1; j11 < i; j11++){
                    if (main_matrix[timeNext][j11][1] == main_matrix[timeNext][i][1] &&
main_matrix[timeNext][j11][2] == main_matrix[timeNext][i][2]){
                                main_matrix[timeNext][i][1] = main_matrix[time][i][1];
                                main_matrix[timeNext][i][2] = main_matrix[time][i][2];
                                main_matrix[timeNext][i][4] = 0;
                    }
                        }
                }
                    }
                }
            }
          }
        }
     i++;
     if(main_matrix[time][i][0] == 0||i>pedestrian_count){
         flag1 = 0;
      }
   }
   averagespeed.close();
   return main_matrix;
}
//-----------------------------------------------------------------------------------------------------------------
------------
public static void main(String[] args) throws IOException {
```

```
        // Main function to execute all the function in a sequential manner and to write outputs in desired format

    System.out.println("       working ... ");

    double[] p = {power(0,1), power(1,1), power(2,2), power(3,6), power(4,24), power(5,120)};// Considering poisons distribution for pedestrian flow

     double ywidth = Math.floor(width / (Math.sin (Math.toRadians(60)) * Tsize));// calculating the units available in Y-direction

    double [][][] main = new double [(total_steps+2)][999][5];              // 3-D array to store pedestrian information. [time step to enter][no. of possible node for movement][ped_ID,X,Y,rand,Velocity]

    ArrayList<Integer> y_pos = new ArrayList<Integer>();                                    // Array list with possible Y units available for movements


    for (int i = 1; i < ywidth; i++){

        y_pos.add(i);

     }

     PrintWriter pedata = new PrintWriter("ped_data.txt");  //steps,ped_count,y_pos

    PrintWriter maindata= new PrintWriter("main_data.txt");  // saving pedestrian data: ped_count, X, Y, random, V

    maindata.println("Step" + "\t"+"ped_id" + "\t"+ "X" + "\t"+ "Y" + "\t"+ "\t"+ "\t"+ "v");

          // Following DO-WHILE loop generates pedestrians

    do {

        double rand = Math.random();                        // generating random number

        Collections.shuffle(y_pos);

        if (rand>p[1]){

          }

        else if (rand>p[2]){
```

```
            pedestrian_count++;

            pedata.println(steps +"\t"+ pedestrian_count+"\t"+ y_pos.get(0));

            main[steps][pedestrian_count][0]=pedestrian_count;

            main[steps][pedestrian_count][1]=(y_pos.get(0)% 2)*0.5;

            main[steps][pedestrian_count][2]=y_pos.get(0);

            main[steps][pedestrian_count][3]=Math.random();

            main[steps][pedestrian_count][4]=vmax;

            maindata.println(steps+"\t"+main[steps][pedestrian_count][0] +"\t"
+"\t"+main[steps][pedestrian_count][1] +"\t"+main[steps][pedestrian_count][2] +"\t"+
main[steps][pedestrian_count][4]);

        }

        else if (rand>p[3]){

            for (int j=0;j<2;j++){

            pedestrian_count++;

            pedata.println(steps +"\t"+ pedestrian_count+"\t"+ y_pos.get(j));

            main[steps][pedestrian_count][0]=pedestrian_count;

            main[steps][pedestrian_count][1]=(y_pos.get(j)% 2)*0.5;

            main[steps][pedestrian_count][2]=y_pos.get(j);

            main[steps][pedestrian_count][3]=Math.random();

            main[steps][pedestrian_count][4]=vmax;

            maindata.println(steps+"\t"+main[steps][pedestrian_count][0]+"\t"
+"\t"+main[steps][pedestrian_count][1] +"\t"+main[steps][pedestrian_count][2] +"\t"+
main[steps][pedestrian_count][4]);

              }

        }

        else if (rand>p[4]){
```

```
            for (int j=0;j<3;j++){
                pedestrian_count++;
                pedata.println(steps +"\t"+ pedestrian_count+"\t"+ y_pos.get(j));
                main[steps][pedestrian_count][0]=pedestrian_count;
                main[steps][pedestrian_count][1]=(y_pos.get(j)% 2)*0.5;
                main[steps][pedestrian_count][2]=y_pos.get(j);
                main[steps][pedestrian_count][3]=Math.random();
                main[steps][pedestrian_count][4]=vmax;
                maindata.println(steps+"\t"+main[steps][pedestrian_count][0]
+"\t"+"\t"+main[steps][pedestrian_count][1] +"\t"+main[steps][pedestrian_count][2] +"\t"+
main[steps][pedestrian_count][4]);
            }
          }
          else if (rand>p[5]){
                for (int j=0;j<4;j++){
                pedestrian_count++;
                pedata.println(steps +"\t"+ pedestrian_count+"\t"+ y_pos.get(j));
                main[steps][pedestrian_count][0]=pedestrian_count;
                main[steps][pedestrian_count][1]=(y_pos.get(j)% 2)*0.5;
                main[steps][pedestrian_count][2]=y_pos.get(j);
                main[steps][pedestrian_count][3]=Math.random();
                main[steps][pedestrian_count][4]=vmax;
                maindata.println(steps+"\t"+main[steps][pedestrian_count][0]
+"\t"+"\t"+main[steps][pedestrian_count][1] +"\t"+main[steps][pedestrian_count][2] +"\t"+
main[steps][pedestrian_count][4]);
            }
```

```
        }
        else{
            for (int j=0;j<5;j++){
            pedata.println(steps +"\t"+ pedestrian_count+"\t"+ y_pos.get(j));
            main[steps][pedestrian_count][0]=pedestrian_count;
            main[steps][pedestrian_count][1]=(y_pos.get(j)% 2)*0.5;
            main[steps][pedestrian_count][2]=y_pos.get(j);
            main[steps][pedestrian_count][3]=Math.random();
            main[steps][pedestrian_count][4]=vmax;
            maindata.println(steps+"\t"+main[steps][pedestrian_count][0]
+"\t"+"\t"+main[steps][pedestrian_count][1] +"\t"+main[steps][pedestrian_count][2] +"\t"+
main[steps][pedestrian_count][4]);
            }
        }
           steps = steps + time_step ;
    }
     while(steps<total_steps);
    System.out.println("Total pedestrians = " + pedestrian_count );
    pedata.close();
    maindata.close();
//-------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
double [][][] int_mat = new double [pedestrian_count+1][999][3];
double [][] [] bas_des_mat  = new double [1+pedestrian_count][98][3];
double [][][][] returnarray = new double [2][][][];
```

```
int[] interesting_pedid={1,6,7,8,9,10,11,12,13,14,15,16};


int countPedData = 12;

PrintWriter[] finalAllPedData = new PrintWriter[countPedData]; // saving pedestrian data:
step, X, Y

for(int i=0;i<countPedData;i++) {

   // finalPedData[1] should write to file finalmain_data2.txt.

   finalAllPedData[i] = new PrintWriter("finalmain_data" + String.valueOf(i+1) + ".txt");

   finalAllPedData[i].println("Data for pedestrian " + interesting_pedid[i]);

   finalAllPedData[i].println();

   finalAllPedData[i].println("Step" +"\t"+ "X" +"\t"+ "Y"+"\t"+ "Velocity");

}

PrintWriter finalpeddata  = new PrintWriter("finalmain_data.txt");

finalpeddata.println(pedestrian_count);

finalpeddata.println("Data for all pedestrian "+"\t"+"lane width"+"\t"+ width +"\t"+
"Flow"+"\t"+ ped_flow+"\t"+"and size
is"+"\t"+Tsize+"\t"+"duration"+"\t"+total_steps+"\t"+"sec");

finalpeddata.println();

finalpeddata.println("Step" +"\t"+ "X" +"\t"+ "Y"+"\t"+ "Velocity");

   timeNext=1;

   steps=0;

   do {

      int_mat = intmat(vmax, int_mat, pedestrian_count);

      bas_des_mat = bdesmat(vmax, bas_des_mat, pedestrian_count);

      for (int j = 1; j <= pedestrian_count; j++) {
```

```
            for (int j2 = 0; j2 < 5; j2++) {
                main[timeNext][j][j2] += main[steps][j][j2];
            }
        }
        for (int i = 0; i < countPedData; i++) {
            finalAllPedData[i].println(steps + "\t" + main[timeNext][interesting_pedid[i]][1] + "\t"
+ main[timeNext][interesting_pedid[i]][2] + "\t" + main[timeNext][interesting_pedid[i]][4]);
        }
        for (int i1 = 1; i1 < pedestrian_count; i1++){
            finalpeddata.println(steps + "\t" + main[timeNext][i1][1] + "\t" +
main[timeNext][i1][2] + "\t" + main[timeNext][i1][4]);
        }
        returnarray = negeff(returnarray, pedestrian_count, total_steps, bas_des_mat,
int_mat, main, steps, vmax, ywidth);
        bas_des_mat = returnarray[0];
        main = returnarray[1];
        if (pedestrian_count >= 1){
            if (steps >= 3 && steps <= 7){
                for (int j = 0; j <= base_des_count; j++) {
                    bas_des_mat[1][j][2] = -100;
                }
            }
        }
        if (pedestrian_count >= 10) {
            if (steps >= 10 && steps <= 12) {
```

```
            for (int j = 0; j <= base_des_count; j++) {
                bas_des_mat[10][j][2] = -100;
            }
        }
    }


        if (pedestrian_count >= 16) {
            if (steps >= 12) {
                for (int j = 0; j <= base_des_count; j++) {
                    bas_des_mat[16][j][2] = -100;
                }
            }
        }
        main = updatePosition(ywidth, pedestrian_count, main, timeNext, steps,
bas_des_mat);
        if ((total_steps - steps) % 10 == 0) {
            System.out.println(total_steps - steps);
        }
        steps++;
        timeNext++;
    }
    while(steps <= total_steps);
        for (int i=0;i<countPedData;i++) {
            finalAllPedData[i].close();
```

```
        }
        finalpeddata.close();
        System.out.println( "\n" + "     "  + "DONE");
        System.out.println( "\n" + "    " + "\\"+"(^.^)/");
    }

    private static double[][][][] negeff(float[][][][] returnarray, int ped_count, int total_steps, float[][][] bas_des_mat1, float[][][] int_mat, double[][][] bas_des_mat, int steps, double vmax, double ywidth) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    private static float[][][] intmat(double vmax, float[][][] int_mat, int ped_count) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    private static float[][][] bdesmat(double vmax, float[][][] bas_des_mat, int ped_count) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
}
```

2. Graphics Code

```
package pedestrian_flow;


import java.awt.*;

import javax.swing.*;


import java.awt.event.*;

import java.io.*;

import java.util.*;

import java.util.concurrent.*;



public class graphics extends JPanel implements ActionListener {


   //-------------------------------------------------------------------------------------------------------------------
-------------

   static int time_gap = 1000;       // in milliseconds


   private static final JTextArea TEXT_AREA1 = new JTextArea();

   private static final JTextArea TEXT_AREA2 = new JTextArea();

   private static final JButton BUTTON = new JButton("START");

   private static final JButton EXIT = new JButton("EXIT");
```

```
    private static int archakClicks = 0;
    private static boolean flag = false;
    static ScheduledFuture<?> currentTask;
    public static BufferedReader reader;

    private static ArrayList<JLabel> labelList = new ArrayList<JLabel>();
    private static int pedCount = 0; // Default value

    private static final int fontSize = 15;
//-----------------------------------------------------------------------------------------------------------------
-----------

    public static void setLabelList() {
        for (int i = 0; i < pedCount; i++) {
            labelList.add(new JLabel(new ImageIcon("pedestrian.jpg")));
        }
    }
//-----------------------------------------------------------------------------------------------------------------
-----------
    public graphics() {
        setLabelList();
        TEXT_AREA1.setLineWrap(true);
        TEXT_AREA1.setWrapStyleWord(true);
        TEXT_AREA1.setEditable(false);
```

```
TEXT_AREA2.setLineWrap(true);

TEXT_AREA2.setWrapStyleWord(true);

TEXT_AREA2.setEditable(false);


JLabel jlbLabel1 = new JLabel(new ImageIcon("testpic.jpg"));


setLayout(null);

int startX = 2;

int startY = 400;

int text1_width = 400;

int text2_width = 400;

int currentX = startX;


TEXT_AREA1.setBounds(currentX, startY, text1_width, 30); // (x,y,width,height)

currentX += text1_width;

TEXT_AREA1.setText("Click START to run simulation");

TEXT_AREA1.setForeground(Color.lightGray);

TEXT_AREA1.setBackground(Color.BLACK);

TEXT_AREA1.setFont(new Font("Courier", Font.BOLD, fontSize));


TEXT_AREA2.setBounds(currentX, startY, text2_width, 30); // (x,y,width,height)

currentX += text2_width;

TEXT_AREA2.setText("Click EXIT to quit");

TEXT_AREA2.setForeground(Color.cyan);
```

```
TEXT_AREA2.setBackground(Color.BLACK);
TEXT_AREA2.setFont(new Font("Courier", Font.BOLD, fontSize));

int buttonWidth = 90;
BUTTON.setBounds(currentX, startY, buttonWidth, 30); // (x,y,width,height)
currentX += buttonWidth;

EXIT.setBounds(currentX, startY, buttonWidth, 30); // (x,y,width,height)
currentX += buttonWidth;
jlbLabel1.setBounds(-85,0,1200,400);
for (int i = 0; i < pedCount; i++) {
   // System.out.println(i);
   labelList.get(i).setBounds(-10,-10,10,10); // (x,y,width,height)
   add(labelList.get(i));
}

add(TEXT_AREA1);
add(BUTTON);
add(EXIT);
add(TEXT_AREA2);
add(jlbLabel1);

BUTTON.addActionListener(this);
```

```
        EXIT.addActionListener(this);
    }

    //-------------------------------------------------------------------------------------------------------------
-------------

    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == BUTTON) archakClicks++;

        if (e.getSource() == EXIT) {
            System.exit(0);
        }

        if (archakClicks % 2 == 1) {
            flag = false;
            BUTTON.setText("STOP");
            startProject();
        } else {
            flag = true;
            try {
                reader.close();
            }
            catch (IOException e1) {}
            BUTTON.setText("START");
```

```
            stopProject();
        }
    }

    //-------------------------------------------------------------------------------------------------------------------
-------------

    public static Future<?> getFuture() {
        return currentTask;
    }

    //-------------------------------------------------------------------------------------------------------------------
-------------

    public static void startProject() {
        try {
            reader = new BufferedReader(new FileReader("finalmain_data.txt"));
            TEXT_AREA1.setText("");
            ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);
            currentTask = executor.scheduleAtFixedRate(new Runnable() {
                String line = "";
                @Override
                public void run() {
                    try {
                        if (flag) {
```

```
                getFuture().cancel(true);
            } else {
                for (int i = 0; i < 4; i++) {
                    reader.readLine();
                }

                for (int i = 0; i < pedCount; i++) {
                    line = reader.readLine();
                    String[] array = line.split("\t");

                    int x = (int)Float.parseFloat(array[1])*20; //*100;
                    int y = (int)(Float.parseFloat(array[2])*Math.sin (Math.toRadians(60))*20);
//*100;

                    labelList.get(i).setBounds(x,y,10,10);
                }

                TEXT_AREA1.setText("Click to STOP to stop");
                TEXT_AREA1.setForeground(Color.red);
                TEXT_AREA1.setFont(new Font("Courier", Font.BOLD, fontSize));
            }
        } catch(Exception e) {
            e.printStackTrace();
```

```
                }
            }
        }, 0, time_gap , TimeUnit.MILLISECONDS);
    } catch (Exception e) {
        try {
            reader.close();
        }
        catch (IOException e1) {}
    }

}

//-------------------------------------------------------------------------------------------------------------------
-------------

public static void stopProject() {
    TEXT_AREA1.setText("Click START to start again");
    TEXT_AREA1.setForeground(Color.green);
    TEXT_AREA1.setFont(new Font("Courier", Font.BOLD, fontSize));

    for (int i = 0; i < pedCount; i++) {
        labelList.get(i).setBounds(-10,-10,10,10);
    }
}
```

```
   //-------------------------------------------------------------------------------------------------------------
-------------

   public static void pedCount() {

      try {

         final BufferedReader reader1 = new BufferedReader(new
FileReader("finalmain_data.txt"));

         String line = reader1.readLine();

         String[] array = line.split("\t");

         pedCount = Integer.parseInt(array[0]);

         reader1.close();

      } catch(Exception e) {

         System.out.println("Error in reading/parsing file.");

      }

   }

   //-------------------------------------------------------------------------------------------------------------
-------------


   public static void main(String[] args) {

      pedCount();

      graphics archak = new graphics();

      final JFrame frame = new JFrame("Footpath");

      frame.setContentPane(archak);

      frame.setVisible(true);

      frame.setSize(1000,500);  // (width,height)
```

```
        frame.setResizable(false);


        frame.addWindowListener(
                new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {
                        System.exit(0);
                    }
                });
    }
}
```