CrowdEgress: A Multi-Agent Simulation Platform for Pedestrian Crowd

Peng Wang Xiaoda Wang Peter Luh Neal Olderman

Christian Wilkie Timo Korhonen Kerry Marsh

This manual introduces a simulation tool to study complex crowd behavior in social context. The agent-based model is extended based on the well-known social force model, and it mainly describes how agents interact with each other, and also with surrounding facilities such as walls, doors and exits. The simulation platform is compatible to FDS+Evac, and the input data in FDS+Evac could be imported into our simulation platform to create single-floor compartment geometry, and a flow solver is used to generate the road map towards exits. Most importantly, we plan to integrate advanced social and psychological theory into our simulation platform, especially investigating human behavior in emergency evacuation, such as pre-evacuation behavior, exit-selection activities, social group and herding effect and so forth.

1. Introduction

The agent-based model (ABM) is a computational research method to study social systems. This model-driven approach partly origins from statistical physics, investigating how individuals in free space or in lattice interact with each other and whether there is any converged pattern emerged at the macroscopic level. Since a society composed of many people is a typical system of many-particle, it is possible to apply the principles of statistical physics to study social behavior of many individuals. Recently, there has been a growing interest in using agent-based model and simulation to understand social phenomena such as economic market or political opinions [Peralta, Kertesz, Iniguez, 2022; Quang et. al., 2018].

The agent-based model refers to a system of many-particle that exhibits emergent characteristics when autonomous agents interact with each other. Basically, the ABM consists of agent, system space, and external environment. The agent is autonomous and decides his/her behavior by interacting with the neighbors or the external environment with the rules of behavior. In our simulation platform, for example, the system space is a 2D planar space, and the environment is given as a structural layout that consists of obstructions (e.g., walls) and passageways (e.g., doors or exits), and other environmental stimuli may be imported such as gas temperature or smoke density in the future. Agents are interacting with each other and moving within this structural layout.

The simulation is mainly implemented by a component as packed in a class called simulation class (simulation.py), and it computes interaction of agents with surrounding entities including walls, doors and exits. The agent model is described in agent.py, whereas walls, doors and exits are coded in obst.py. The agent-based model is an extension of the well-known social force model (Helbing and Monlar 1995; Helbing, Farkas, Vicsek 2000; Lakoba, Kaup and Finkelstein 2005). This model has been applied in many existing pedestrian simulators such as PTV Viswalk, MassMotion, FDS+Evac and so forth [Santos and Aguirre 2005; Ronchi, R. Lovreglio, M. Kinsey, 2020]. The model aims at investigating protypes of pedestrian behavior in crowd evacuation. The core algorithm is still being developed and improved. This is an inter-discipline study topic, which refers to Newton particles, statistical physics,

dynamic systems as well as social and behavioral science. Your contributions or comments are much welcome. The program source code and numerical test cases are mainly uploaded online at https://sourceforge.net/p/crowdegress/code

The program also consists of several functional components such as User Interface and Data/Visualization Tool.

***User Interface**: The user interface is written in tkinter in ui.py. Please run ui.py to enable a graphic user interface (GUI) where one selects the input files, initialize compartment geometry, and configure or start a simulation. An alternative method is using main.py to directly start a simulation without GUI. Currently there is a simple version of GUI and it needs to be improved in several aspects.*

***Data Tool**: This component is packed in data_func.py, and it reads in data from input files, and write data to output files. The input data is written by users in either csv files or fds input files [McGrattan et. al., 2021]. Agents must be specified in csv file while walls, doors or exits can be described either in csv file or read from standard fds input file. The simulation output is written into a binary file, which is compatible to the fds output data (fds prt5 data format). In the future we plan to visualize such data by smokeview [Forney, 2022], which is the standard tool to visualize fds output data.*

***Visualization Tool**: The visualization component is packed in draw_func.py and currently pygame (SDL for python) is used to develop this component. Users can select to visualize the simulation as it runs, or visualize the output data after the simulation is complete. If anyone is interested, please feel free to extend the module or try other graphic libraries to write a visulization component.*

2. About Simulation Models

Agent-based model (ABM) describes interactions among individual agents and their surroundings. In the simulation there are four types of entities: agents, walls, doors and exits. As below we will introduce how to specify these entities.

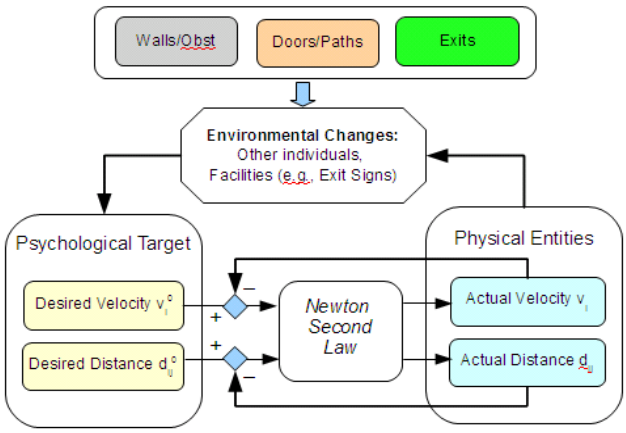
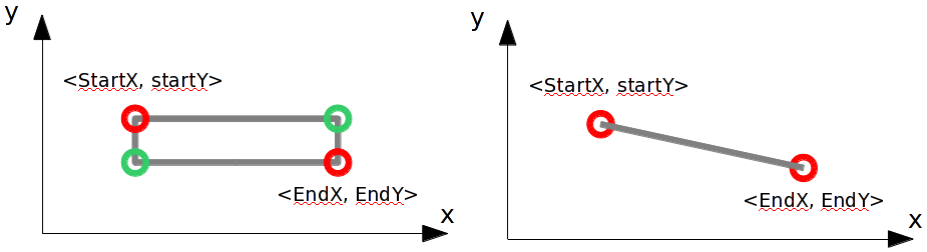


Figure 1. Agent-based model with Environmental Facilities.

The walls, doors and exits are alternatively specified by FDS input files. Users are welcome to use existing FDS input files to create compartment geometries. In current version only one-floor crowd simulation is supported. So if there are multiple evacuation meshes in FDS input files, they should all belong to the same z interval in the vertical direction (z axis). By using FDS input files the walls are created by &OBST, and the doors are specified by &HOLE or &DOOR. The exits are obtained from &EXIT in FDS input files. If users want to find more about how FDS define a compartment area, please refer to FDS UserGuide for more information [McGrattan et. al., 2021]. If users do not use FDS input files, the above entities can alternatively be specified by using csv files as introduced below, and users need to write data blocks in csv file, and such data blocks are identified by &Door, &Exit, &Wall and &Ped as the first element in the data block.

Walls: *Walls are obstruction in a compartment geometry that confine agent movement, and they set up the boundary of a room or certain space that agents cannot go through. &Wall is the identifier for the data block of walls. In other words, &Wall claims that this data block describes walls in the simulation, and thus &Wall should be written as the first element in the data block, namely, the most upper left element in the array-like data sheet. Please refer to examples for more details.*

In our program walls are either lines or rectangular areas. If any users are interested, please feel free to extend the wall types to circular or polyangular areas. If users import walls from a FDS input file, the walls are created as a rectangular type and it corresponds to &OBST in FDS input file. If users specify a line obstruction, it is expected to input the position of starting point and ending point of a line. If users specify a rectangular obstruction, it is expected to input the diagonal position of two points to shape a rectangular area.



(a) Rectangular Type *(b) Line Type*

Figure 2. Create Walls in Rectangular Type or Line Type.

<name>: Name assigned to the wall, and it is arbitrarily given by users. Leave it blank if no name is assgined

<startX, startY>: One diagonal point for rectangular obstruction; Or starting point for line obstruction.

<endX, endY>: The other diagonal point for rectangular obstruction; Or ending point for line obstruction.

<direction>: Direction assigned to the obstruction so that agents will be guided when seeing this obstruction, especially when they do not have any target door or exit. The direction means if the obstruction provides agents with any egress information or not (e.g., exit signs on the walls which direct agents to the location of exits). The value could be +1 for positive x direction, -1 for negative x direction, +2 for positive y direction and -2 for negative y direction. If no direction is given, the value is 0 by default, which means the obstruction does not provide any information of egress or exit locations. User may optionally leave it blank such that the default value 0 is used.

<shape>: Either rectangular or line obstruction in current program. The default value it 'rect', which means the rectangular shape. Use string 'line' if the obstruction is in line shape.

Table1. Data Array of Wall

&Wall	*1/startX*	*2/startY*	*3/endX*	*4/endY*	*5/direction*	*6/shape*
Wall Down	*0*	*0*	*10*	*0.5*	*0*	*rect*
Wall Top	*0*	*9.5*	*10*	*10*	*0*	*rect*
WallLeft	*0*	*0*	*0.5*	*10*	*0*	*rect*
WallRight	*9.5*	*0*	*10*	*10*	*0*	*rect*

Doors and Exit: *Doors are passageways that direct agents toward certain areas, and they may be placed over a wall so that agents can get through the wall by the door. Doors can also be placed as a waypoint if not attached to any walls, and they can be considered as arrows or markers on the ground that guide agent egress movement. In brief doors affect agent way-finding activities and they help agents to form a roadmap to exits. In current program doors are only specified as rectangular object, and the data block of doors is claimed by &Door, which should be written as the first element in the data block. Please refer to examples for more details.*

Exits are a special types of doors which represent paths to the safety. Thus they may be deemed as final path to reach safety areas, and computation of an agent is complete when the agent reaches an exit. An exit is usually placed over a wall like doors, but it can also be put anywhere independently without walls. In the program exits are only defined as rectangular areas. The data block of exits are claimed by &Exit, which should be written as the first element in the data block. Please refer to examples for more details. The specific features of doors and exits are given as below.

<name>: Name assigned to the door/exit, and it is arbitrarily given by users and optionally visualized in the simulation window (pygame screen). Users could identify each door or exit by their names assigned in the input file. If no name is assigned, please leave the cell blank in the input csv file.

<startX, startY>: One diagonal point for rectangular door/ exit.

<endX, endY>: The other diagonal point for rectangular door/exit.

<direction>: Direction assigned to the door or exit so that agents will be guided when seeing this entity, especially when they do not have any target door or exit. The direction implies if the door or exit provides evacuees with any egress information such as exit signs or not. The value could be +1 for positive x direction, -1 for negative x direction, +2 for positive y direction and -2 for negative y direction. If no direction is given, the value is 0 by default, which means the door/exit does not provide any information of egress or exit locations. The direction is partly used in the way-finding algorithm when agents adapt their routes to exits. If users do not know how to specify the direction, simply leave it blank such that the default value 0 is used. Please refer to FDS+Evac manual to better understand the direction setting.

<shape>: All the doors/exits are only in rectangular shape in current program. The default value it 'rect', which means the rectangular shape.

Table2. Data Array of Door

&Door	1/startX	2/startY	3/endX	4/endY	5/direction	6/shape
Door Down	4.49	0	5.51	0.6	0	rect
Door Top	4.49	9.4	5.51	10.1	0	rect

Table3. Data Array of Exit

&Exit	1/startX	2/startY	3/endX	4/endY	5/direction	6/shape
Exit Down	4.5	−0.3	5.5	0.3	0	rect
Exit Top	4.5	9.7	5.5	10.3	0	rect
Exit Right	9.3	4.3	10.3	5.9	0	rect

***Agents**: Finally and most importantly, agents are the core entity in computation process. They interact with each other to form collective behavior of crowd. They also interact with above types of entities to form egress motion toward exits. The resulting program is essentially a multi-agent simulation of pedestrian crowd. Each agent is modeled by extending the well-known social force model. The model is further advanced by integrating several features including pre-evacuation behavior, group behavior, way-finding behavior and so forth. In current program the data block of agents are claimed by &Ped or &Agent, and they are written as the first element in the data block.*

<name>: Name assigned to each agent and it is arbitrarily given by users and optionally visualized in the simulation window (pygame screen).

<InitalX, InitialY>: Initial position of an agent in 2D planar space.

<InitialVx, InitialVy>: Initial velocity of an agent in 2D planar space.

<tau>: Tau parameter in the social force model, or as usually called relaxation time in many-particle systems or statistical physics, and it critically affects how fast the actual velocity converges to the desired velocity.

<tpre>: Time period for pre-evacuation stage. Within this time period agents do not select and move towards an exit.

<pD>: Parameter p in opinion dynamics, and it indicates how an agent's opinion/decision is impacted by surrounding others, and it critically affects herding effect in collective behavior. The measurement of this parameter is within [0, 1], and an agent's opinion/decision completely follow others if p=1. In contrast, if the agent makes decision only based on his or her own opinion or judgement, the parameter p=0.

<pDMode>: This parameter affects how parameter p is dynamically changing. Currently there are three values to be selected: random, fixed or stress. If 'random' is selected, it means that parameter p is randomly generated in the interval of [0, 1]. If 'fixed' is selected, parameter p is given by the initial value in the input csv file. If 'stress' is selected, then a computational model is used adapt the parameter p dynamically to surroundings.

<pp2>: This parameter affects how much an agent tends to make a decision based on the information timely collected from the srrounding facilities such as the distance to the target exits or received guidance from broadcast. The measurement of this parameter is within [0, 1], and an agent's opinion/decision completely follows the received information if pp2=1. In contrast, if the agent makes decision only based on his or her past experience and the new information is completely ignored, the parameter pp2=0.

<talkRange>: The range to determin when agents have opinion exchange. Such interaction could also be understood as herding effect or group opinion dynamics, which means agents exchange opinions by talking.

<aType>: The type of way-finding behaviors. Some agents may actively search for exits while others may just follow the crowd. In current simulation all agents either follow the egress flow field or find their ways based on solver selected, and thus this parameter is not actually used in existing version of code.

<inComp>: a boolean variable to indicate if the agent is in computation loop or not. Normally it is given true. If users want to quickly remove an agent in computation loop, please assign it false for convenience.

<DestX, DestY>: Destination position in 2D planar space. This value is almost not used in current computational loop because the destination position is automatically determined by the exit selection algorithm. When the exit is selected by an agent, the destination position is given by the exit position.

<mass>: The mass of agents.

<radius>: The radius of agents.

Table4. Data Array of Agents

&Agent	*00/ IniX*	*01/ IniY*	*02/ IniVx*	*03/ IniVy*	*04/ tau*	*05/ tpre*	*6/p*	*7/ pMode*	*8/pp2*	*9/talk Range*	*10/ aType*	*11/ inComp*	*...*
Agent0	*6.3*	*2*	*1.6*	*1.0*	*0.3*	*3*	*0*	*fixed*	*0.7*	*30*	*Active*	*1*	*...*
Tuck	*7.3*	*3*	*0.6*	*0.2*	*0.6*	*2*	*0.6*	*random*	*0.6*	*31*	*Follow*	*1*	*...*
James	*7.3*	*6*	*0.3*	*0.7*	*0.6*	*2*	*0.6*	*random*	*0.6*	*30*	*Search*	*1*	*...*

Important Notice:

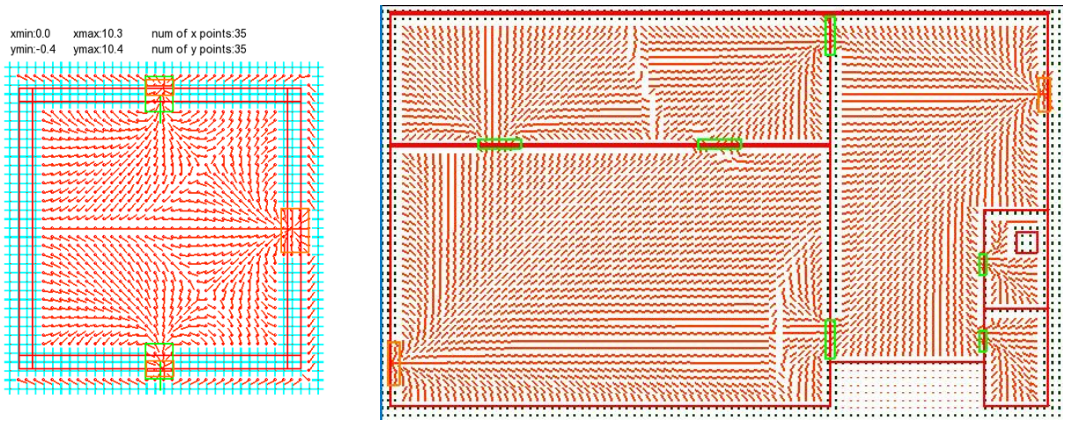
(1) The sequence of &Wall &Door &Exit &Agent could be arbitrarily changed within an input file. Users may either first specify walls, or doors or agents. However, data feature inside a data block could not be changed in sequence. For example, you must first give <startX>, and next <startY> for walls, doors and exit, and cannot change the sequence to be <startY> and <startX>, or move <endX> or <endY> before them. All the features for walls, doors, exits and agents are read in certain order, and such order cannot be altered in current version (version 2.2). New contributors are welcome to improve this functional setting in data_func.py such that this sequence can also be adjusted by users.

(2) As above we exemplify all the input data by table-like data sheet, and users can select any table processing software such as Excel or GNUMeric to show such data. However, please remember that csv (comma separated values) is the basic format of the above data file, namely all the data cell in the input file are separated by comma. Those comma are omitted in Excel or GNUMeric, but are well shown in any text editor.

(3) Please note that all the entities including walls, doors, exits and agents are also assigned with a unique ID number, and this number is automatically given by the program. The ID number starts from zero and it increases based on the sequences of entities specified in the input csv file. Both of the name and ID numbers could be

visualized in simulation window (pygame screen). If users hope to change this ID number, the only method is adjusting the sequence of specifying these entities in the data array of input csv file.

***Exit Selection Probability**: Way finding refers to how individual agents orientate themselves towards exits, especially within a multi-compartment layout. Several factors influence their decision making in way-finding activities such as guidance information (e.g., exit signs) or other individuals' choice.*



(a) *(b)*

Figure 3. Flow field of using the nearest exit.

If users select the default setup of using the nearest-exit strategy, it implies that each agent will select the exit nearest to their current locations, and no exit-selection probability is much involved. This is indicated as solver 1 in our program, and it is the basic setup for a multi-agent simulation.

Another solver deals with a more complicated scenario, where each individual's choice is determined by exit-selection probability distribution. This process is indicated as solver 2 in our program, which can be seen as the result of a hierarchical decision making process entailing two stages: (1) tactical choice of exits and routes; and (2) operational short range choices concerning the interaction with obstacles and other evacuees [Lovreglio, Fonzone, and dell'Olio, 2016]. In other words we suppose that there are several candidate exits known by evacuees and simulate how they select one among these exits as well as choose a proper way to reach them.

The tactical choice is described by a data array as exemplified above, where each row corresponds to each agent and the column refers to exit selection probability. The entire data array is claimed by &Ped2Exit or &Agent2Exit, which is written in the potion of the first element. All the agents select exits base on the discrete probability distribution as described by the array.

Table 5. Data Array of Exit-Selection Probability

&Ped2Exit	*exit0*	*exit1*	*exit2*
agent0	*0.7*	*0.2*	*0.1*
Tuck	*0.3*	*0.3*	*0.1*
James	*0.5*	*0.2*	*0.3*

The operational choice is to generate the evacuation route based on the exit selected. There are a number of methods used in existing egress simulators. Our algorithm is partly learned from FDS+Evac, where the route is calculated by a two-dimensional flow solver. The computation result is a 2D flow field that guides agents to the exit selected. The flow field can be better explained as a social field related to social norms or other behavioral characteristics (Lewin, 1951; Helbing et. al., 2000; Wang, 2022), and we will further elaborate the idea in future. In this algorithm each exit is a sink point, and solver calculates the route as the crowd flow move to the sink (Korhonen et. al., 2008; Korhonen, 2016). The detailed discussion of the flow solver will not be included in this

article, but we emphasize that this method is more suitable to describe human collective behavior on the background of social science and psychological theory.

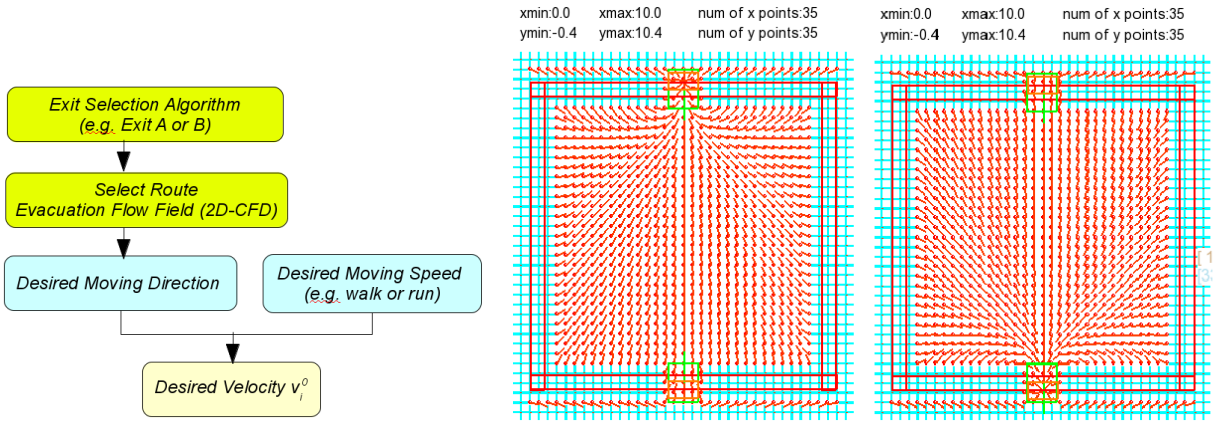


Figure 4. Simulation of crowd egress with exit-selection algorithm.

In the above setting of solver2 the exit-selection probability may be time-invariant for each agent. A more realistic and complicated scenario assumes that each individual's choice is made by integrating three major factors: (1) historical knowledge of building layout or prior information such as habit of using different path; (2) Judgment on current situation in egress (e.g., the distance to an exit) (3) Decisions of other people. Because each individual's decision is interacting with others, the collective decision-making is critically studies, and several social and economic models are extended to describe this process with integration of various social-psychological findings in evacuation research [Ozel, 2001; Staal, 2004; Santos and Aguirre 2005]. To fully describe this scenario users should well define parameter <pD> for each agent in csv file. As a result, the data array claimed by &agent2exit only gives the initial value of exit-selection probability distribution, and <pD> critically determines how it evolves dynamically in the simulation.

3. How-To

As above we briefly introduce how to write csv file to describe agents, walls, doors and exits. In addition another config.txt file is often used jointly with csv file to configure the simulation parameters. In our program there are mainly two methods to configure a simulation object. One way is using GUI, and it is the most user-friendly method to setup all the simulation parameteres. The other one is using a script file, config.txt, which is to be placed in the same folder as the input csv file. In this section we will first explain how to use GUI (tkinter window) to set up the simulation parameters. Writing config.txt is briefly mentioned next. The output data files include a binary data file, a text file and a npz data file (a special form in Numpy to store data array or matrix-like data). Before we move to introduce GUI, there is an important parameter to be mentioned first, namely, the solver for agent-based model.

Table 6. Solver of Agent-Based Model

Solver = 0	*Do no use egress flow field, and agents find their ways by none-flow algorithms. In this solver agents may go to their destination site if no exit is specified in the input file. Currently, this solver has some problems to be solved. So users have no direct access to this solver by using GUI. In brief if no exit is specified in the input file, Solver=0 will be enabled automatically because egress flow field cannot be computed with no exit.*
Solver = 1	*Compute egress flow field for the nearest-exit strategy, and agents follow this flow field to use the nearest-exit to their locations. Agents do not move to exit when t>tpre if no exits is specified in the input file.*
Solver = 2	*Compute egress flow field for all possible exits and agents select one exit based on exit-selection probability. Agents do not move to exit when t>tpre if no exits is specified in the input file.*

In Tkinter GUI:

When tkinter window (GUI) is activated, users will see several panels including <QuickStart>, <Parameters> and so forth. Next, we will briefly introduce how to set up the simulation in the panels.

The default panel is called <QuickStart>. In this panel users select the input files to get a quick run of the simulation. Choose csv file to specify agent data. Users can optionally use fds file to create the compartment geometry, and the pedestrian features are described in csv file. If both csv and fds files are presented, the compartment structure will be created by fds file. If fds file is omitted, the compartment geometry should be described in csv file. The agents must be specified in csv file currently while the walls, doors and exits can either described by csv file or fds file. Please take a look at the examples for details.

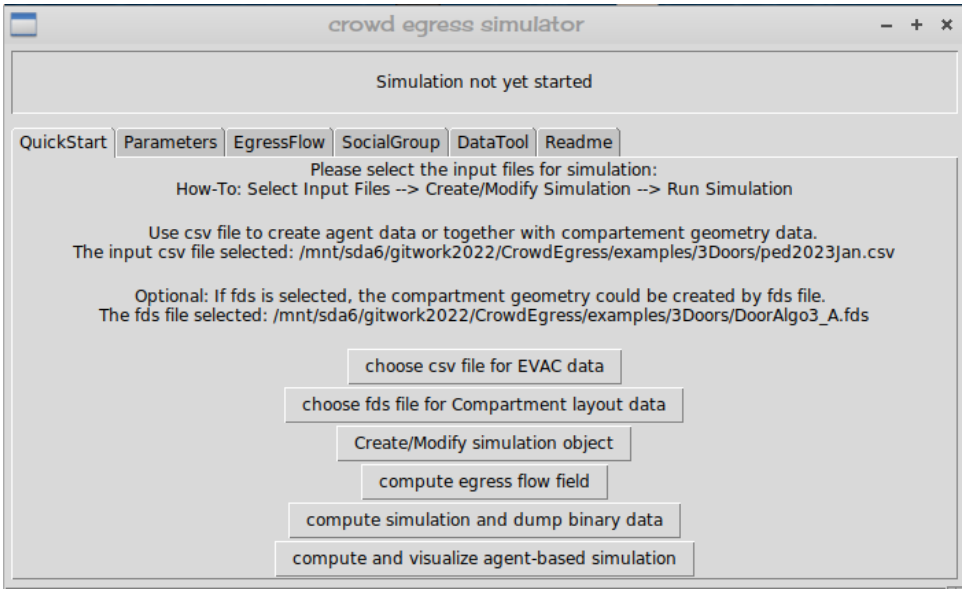


Figure 5. User Interface of CrowdEgress (QuickStart)

<Create/Modify simulation object>: After the input files are well selected, it is suggested that users first create the simulation object to see if the structural layout is created as expected. Another screen is next displayed by pygame, where users could check all the entities in the simulation, such as wall, doors, exits and initial potions of each agent. We call this screen TestGeom in this manual, and it means testing the geometric settings of compartment layout. In TestGeom user could add walls, doors or exits, or change the direction of doors or exits, or change the initial position of each agent. Whenever users change the input files or after an existing simulation is complete, users should re-click this button to initialize the simulation object. Otherwise the simulation object will remain to be the previous one in the memory and new simulation object will not be created.

In Pygame Screen (TestGeom):

When pygame screen is activated, press keys of PageUp/PageDown in your keyboard to zoom in or zoom out the entities in screen. Use direction keys to move the entities vertically or horizonally in screen. In fact there are several sections in pygame screen, and these keys work in similar ways in all the sections. The first section we will introduce is called TestGeom as shown in Figure 6, where users can visualize compartment geometric settings and modify them manually.

In TestGeom users can add walls, doors or exits by selecting the corresponding items in the menu bar (See Figure 7). The direction of a door and exit could also be adjusted by drawing a line from outside to inside of the door or exit. Sometimes it is necessary to adjust the exit-selection probability if new exits are added in TestGeom. Namely, the number of column in data array &Agent2Exit (See Table 1) should be equal to the total number of exits.

Users can also dump geometric data (e.g., wall data and door data) into csv file by selecting the item <OutputData> in the menu bar. The output file is created as bldDataRev.csv for any modification of the compartment geometric in TestGeom. The data can also be briefly shown in the screen by selecting the item

<ShowData> in the menu bar. If users click <Simulation>, then the simulation starts and the program goes to the second section called RunSimulation.

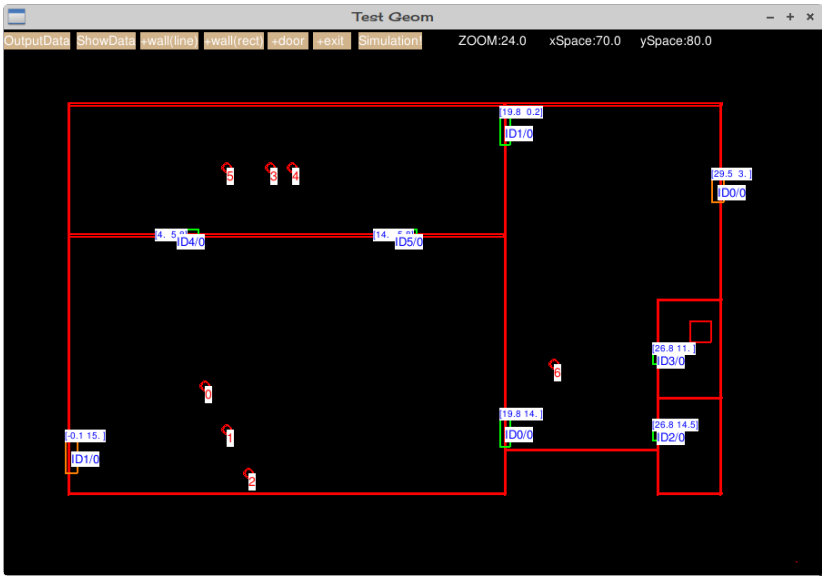


Figure 6. Pygame screen of TestGeom.

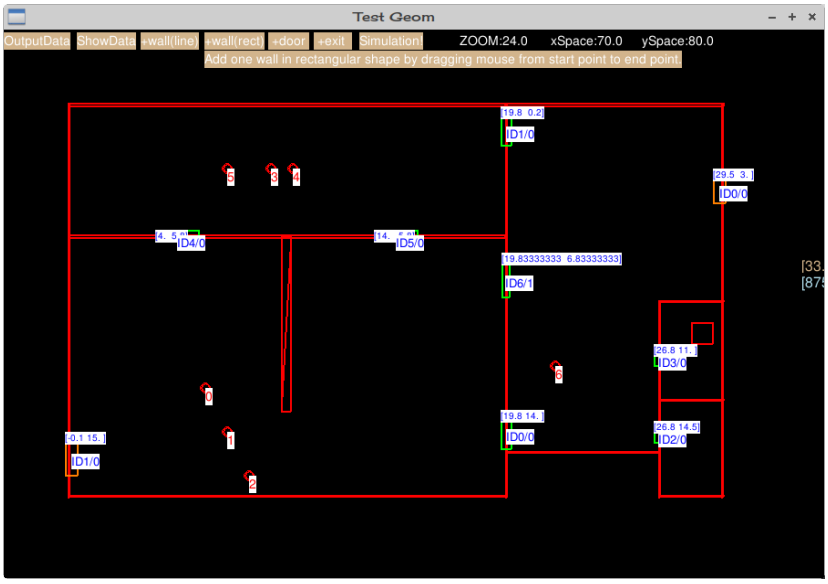


Figure 7. Add walls and doors in TestGeom.

After the simulation object is created or modified, the next step is running the simulation. There are several options for users.

<Compute Egress Flow Field>: The first option is to compute the egress flow field and display it on the pygame screen, and agent-based simulation is not started in this option. In the pygame screen users will find how the mesh is created such as the number of x points and y points, the boundary value of mesh (i.e., min_x, max_x, min_y, max_y). Users will also see the flow field towards each exits. If solver 1 is selected, there is only one flow field computed, which is simply using the nearest-exit strategy (See Figure 3). If solver 2 is selected, there are several flow field computed, which corresponds to the roadmap towards each exit (See Figure 4). The nearest-exit strategy is also additionally computed in solver 2. Users could press <o> and <p> in the keyboard to switch from different flow field. Please also see the third panel of EgressFlow for more details.

<Compute Simulation and Dump Binary Data>: The second option is to compute the agent-based simulation with the flow field. The numerical result will be written into a binary file (.bin), a text file (.txt) and a npz data file (.npz). but not displayed timely in pygame screen. If your input csv file is named by agent.csv and simulation is complete at time of 2024-03-02_23_09_27 (in format of year, month, date, hour, minute and second), the output data files will be generated as agent_2024-03-02_23_09_27.bin, agent_2024-03-02_23_09_27.txt and agent_2024-03-02_23_09_27.npz. We will introduce these files in detail in the following section.

<Compute and Visualize Agent-Based Simulation>: The third option is to compute the agent-based simulation and visualize the result timely in pygame screen. This is the most common and important option such that users are able to directly observe how agents interact and move towards a selected exit in the compartment layout. Users can show the forces or movement trace of each agents, pause the simulation, and we will introduce this issue soon later.

In Pygame Screen (RunSimulation):

The other section is RunSimulation, in which the agent-based simulation is started and visualized on the pygame screen. User can pause the simulation, but cannot rewind it in current version (Version 2.2). A binary data file is optionally created when the simulation runs and users can also visualize the data after the simulation is terminated. The output data files also include a text file and a npz data file, which have the same prefix name as the binary data file. Users could direct open the text file by using any text editors and check the agent data there. As for the npz data file, it is a special form used by numpy package in python, and it is easy to store the data array and matrix data. This matrix-based data could be visualized together with the binary data file in our program.

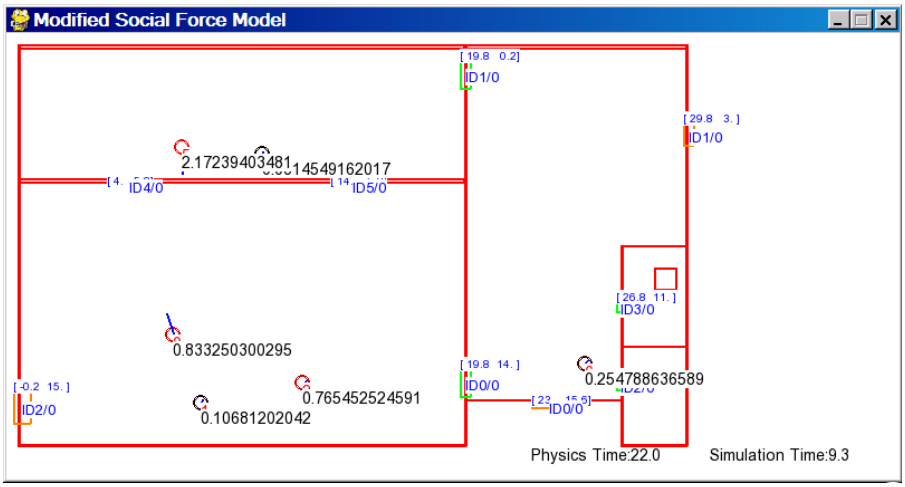


Figure 8. Agents move with stress level indicated.

In both phases of TestGeom and RunSimulation, there are hot keys defined. Use pageup/pagedown to zoom in or zoom out the entities in screen. Use space key to pause the simulation. Use direction keys to move the entities vertically or horizonally in screen. Use 1/2/3 to display the door or exit data on the screen. Press I to show agent index number. Press S to show their stress level timely. The stress level is computed based on our work Wang and Wang 2020.

The second panel is called Parameters, where users specify the basic parameters before the simulation starts. An important issue is that z-interval (min_z and max_z) should be specified when a FDS+Evac input is used to create the compartment geometry setting. The reason is that FDS+Evac is a 3D simulator, and its input file may include several computation meshes as several floors in a building. However, our simulator CrowdEgress is a 2D simulator only for single-floor layout. Thus, if a FDS+Evac input file is used with several compartment floors, sometimes users should check or modify the min_z and max_z to determine which floor should be computed in the simulation. The default value in CrowdEgress is given as min_z=0.0 and max_z=3.0, and it is commonly useful for most FDS+Evac file with only one compartment floor.

Other parameters are briefly introduced as below.

<Show Time in Simulation>: Show the computational time and simulation time when simulation starts.

<Show Stress Level in Simulation>: Indicate the difference between actual velocity and desired velocity. Please refers to Wang, 2021 for more details.

<Use nearest exit strategy>: Solver 1 is used if selected. Namely, each agent is guided to the nearest exit and exit-selection algorithm is not involved. Solver 2 is used if unchecked.

<Show compartment data in simulation>: Show the geometric data of doors and exits in the pygame screen such that users can identify each door and exit by their indice and default directions.

<Show forces on agents in simulation>: Visualize forces in the pygame screen. The forces are measured by several lines shown on each agent. The line in purple color is the wall force; line in green color represents the door force. The line in pink color is the social force.

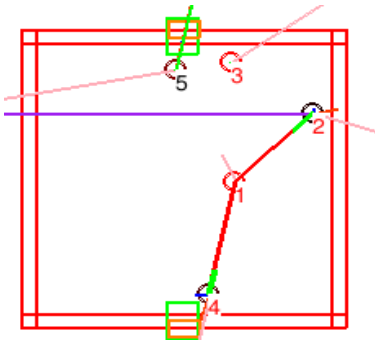


Figure 9. Forces on Agents

<Dump data to a binary file>: Dump simulation data into a binary file which is compatible to fds prt5 data format. The data file is used to visualize the numerical result after simulation stops.

<Group behavior>: Use group social force in simulation. This is a little complicated issue and we will elaborate it in future.

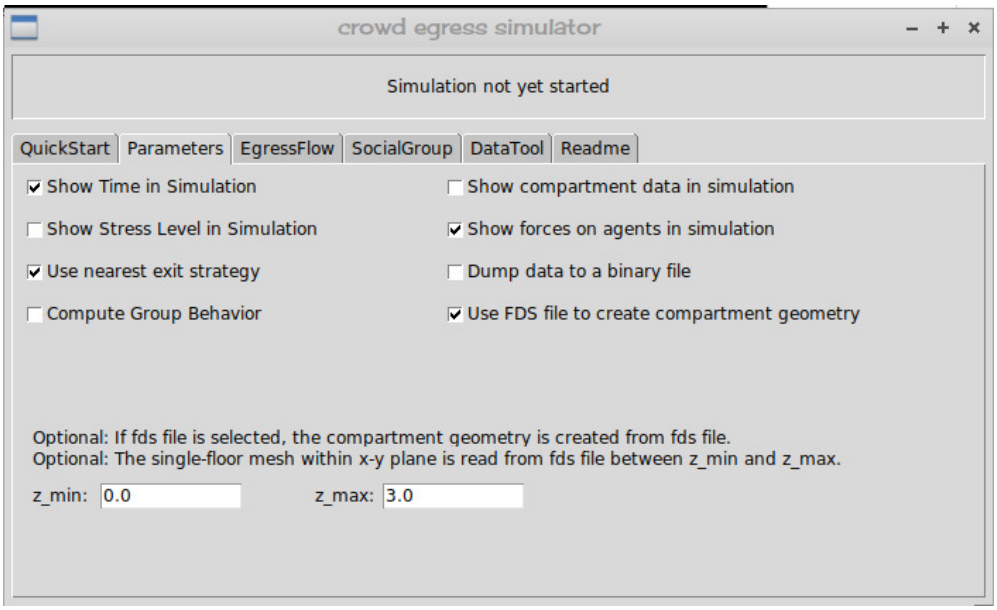


Figure 10. User Interface of CrowdEgress (Parameters)

The third panel is called EgressFlow, where users specify the basic parameters to calculate the egress flow field. The calculation is executed based on the solver selected. If solver=1, the egress flow field is calculated only for the

nearest-exit strategy. If solver=2, several flow fields are calculated, and each one corresponds to an exit, and the total number is the number of exits plus one because the nearest-exit strategy is also calculated for solver=2.

In the panel of EgressFlow users are expected to input the dimensional measure of the calculation mesh, including the minimal and maximal value in x axis (x_min and x_max) and minimal and maximal value in y axis (y_min and y_max). The number of points are denoted by num_x and num_y. When the number of points increases, the computational mesh are refined and the calculation time is supposed to be increased also.

There are two buttons in this panel. One is <Compute Egress Flow Field>, and it is the same as in panel of QuickStart, and it generates a pygame screen, where users will find how the mesh is created such as the number of x points and y points, the boundary value of mesh (i.e., min_x, max_x, min_y, max_y). Users will check the flow fields in the pygame screen: if solver 1 is selected, only one flow mesh is computed for the nearest-exit strategy (See Figure 3). If solver 2 is selected, several flow fields are computed, which corresponds to the roadmap towards each exit (See Figure 4). The nearest-exit strategy is also additionally computed in solver 2. Users could press <o> and <p> in the keyboard to switch from different flow field.

Another button is especially useful for Solver 1 and it shows the computational result of crowd fluid dynamics. This fluid-based model is different from the agent-based model, and it assumes crowd movement as mass flowing in a two-dimensional space, and it is basically an analytical model at the macroscopic level, aggregating many particle-like individuals into fluid-like model of crowd. In a sense the fluid-based model (macro-level) is derived from homogeneous individual equation (micro-level) based on physics laws and *mathematical principles. The resulting model is a set of partial differential equations (PDE). The analytical solutions to these equations are often difficult in mathematics, but numerical solutions are obtained nowadays by advanced computing methods. Thus, we apply a simple version of fluid-based model in our program by extending 1D traffic problem (Lighthill and Whitham, 1955) to 2D crowd fluid problem. The computation process of this simple fluid-based model is included in Solver 1. In other words when users select <Use nearest exit strategy> in the <Parameters> panel, it means that Solver 1 is selected and this fluid-based model will be calculated along with the egress flow field.*

The solution could be visualized by using the second button in <EgressFlow>, i.e.,the button named by <Read output npz file and show crowd fluid model>. The computational result is stored in vel_flow1.npz in the example folder, and it is visualized in Figure 9.

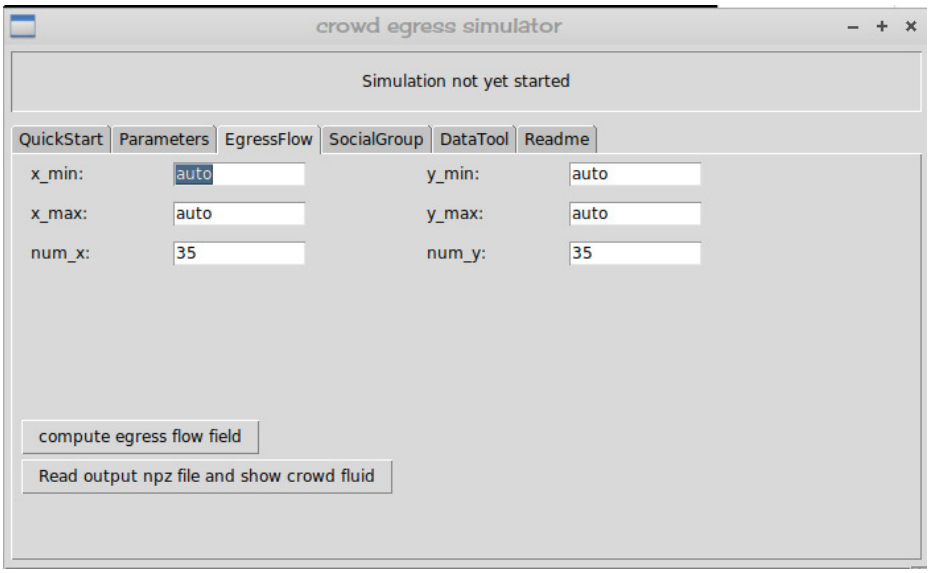


Figure 11. User Interface of CrowdEgress (EgressFlow)

As shown in Figure 9 the basic information of egress flow field is first displayed such that users could check if the mesh is generated as expected, including the number of x points and y points and the boundary value of mesh (i.e., min_x, max_x, min_y, max_y). When the pygame screen is next displayed, the crowd flow density is denoted by a small black dash in the mesh, and users will see how they move towards the nearest exit. In this process it is also easy to use mouse to check the numerical value of the flow velocity and density for each mesh unit. U and V are the flow velocity, and R represents the flow density. Ud and Vd are the desired flow velocity, which always point towards the nearest exit. BLD means whether the mesh unit is free for crowd flowing or solid boundary generated from the compartment layout.

In this process users could also use pageup/pagedown to zoom in or zoom out the entities in screen. Use space key to pause the simulation. Use arrows to move the entities vertically or horizonally in screen.

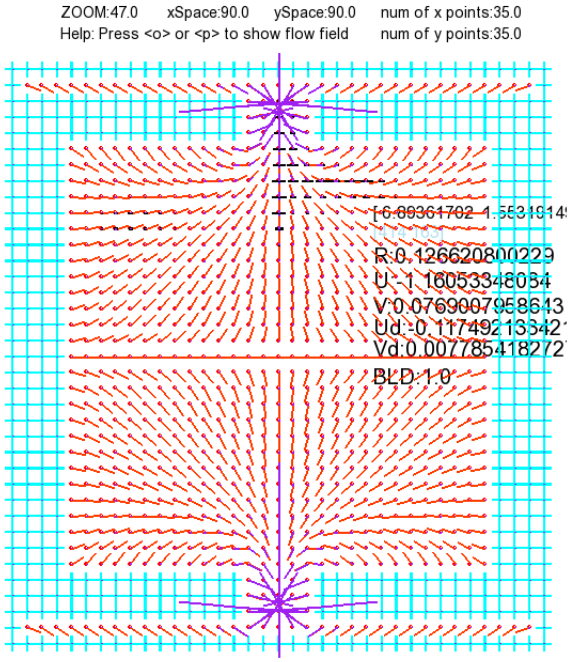


Figure 12. Computational Result of Crowd Fluid Model

Last but not least, analog of fluid dynamics is only valid to crowd at medium or high density, where continuity hypothesis holds for crowd flowing in a planar space. If there are only sparse individuals, continuity hypothesis cannot hold, and fluid-based analysis is not actually suitable for such low-density crowd. The resulting fluid model provides a practical perspective to explain crowd behavior at bottlenecks (e.g., narrow passages), where crowd density are sufficiently large and short-range physical interaction among people plays an important role. Such interactions are among the major cause of crowd disastrous events like stampede.

The general framework of the fluid-based analysis is illustrated as below.

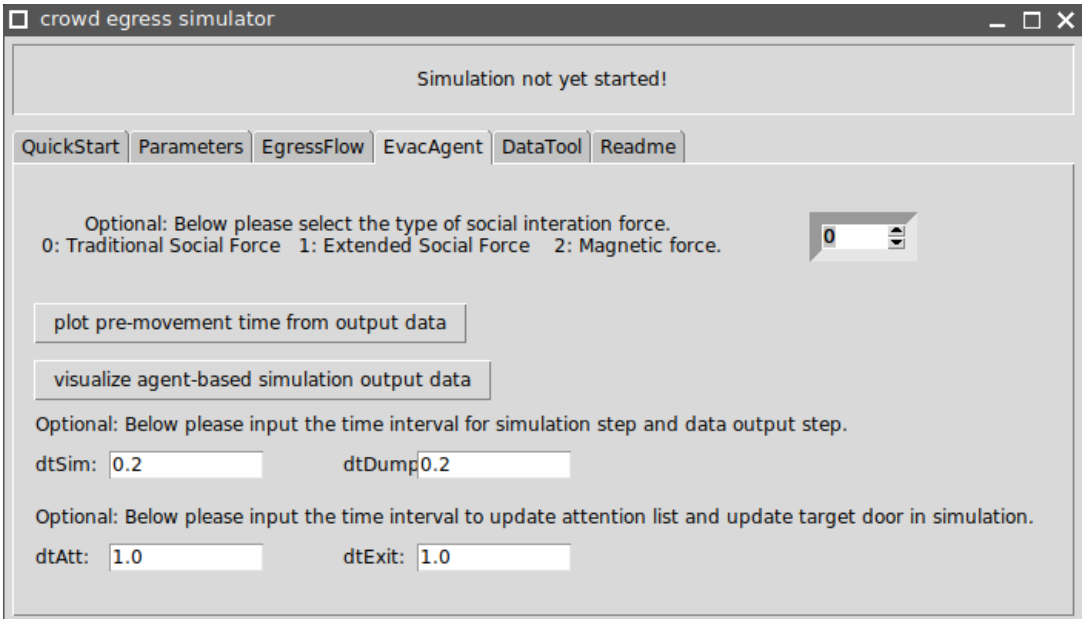


Figure 13

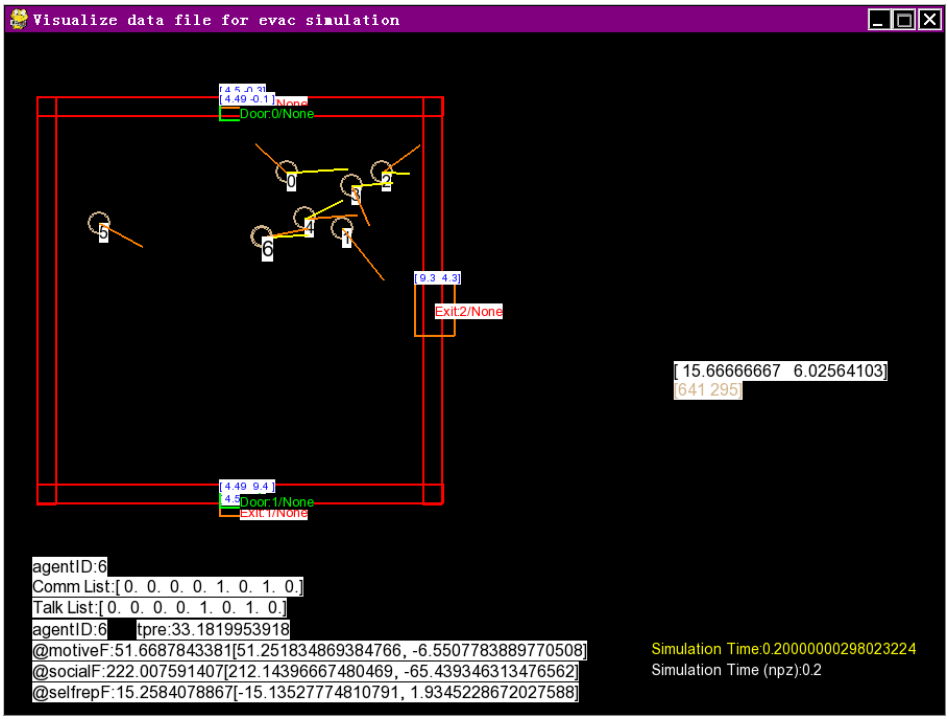


Figure 14. Visualization of simulation output data

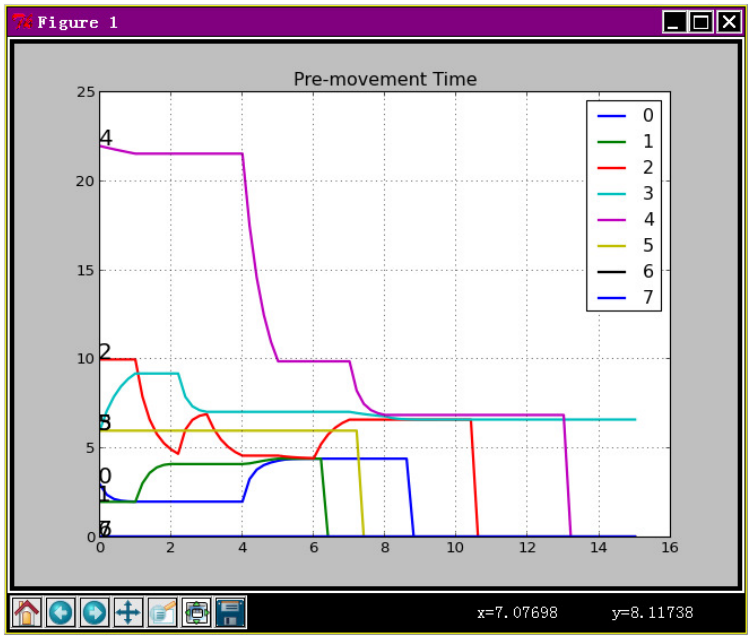


Figure 14. Visualization of pre-movement time from output data

There are several other panels including Fluctuations, DataTool, and the GUI is under development. Thus, we will introduce them in the future.

Output Data:

The output data of above agent-based simulation are classified in two forms, namely, the non-matrix data and matrix-based data. The non-matrix data is adpoted to store agents' position velecity and forces for all the time steps. The social relationship and egress flow field are matrix-based data in our program. Both forms are also written into a text file in the simulation process. Users could easily check the text data by a text editors. The output data files are listed as below.

Table 7. Output Data of Simulation.

Text Data	*The text file is a log of computation process and it records agents' position velecity and forces for all the time steps. It also includes other agent features such as exit selection probability, dynamics of pre-movement time, the matrices of social relationship, and so forth. In a sentse the text file contains all the data, which are included in the binary file and NPZ file.*
Binary Data	*The binary data file mainly records agents' position velecity and forces for all the time steps. It also record other non-matrix data for each agents, such as pre-movement time, stress level, exit selected and so forth. The binary data could be visualized in pygame after the simulation is complete. Use a button in the panel of EvacAgent and select the binary file (suffix of .bin file), and the data will be extracted and displayed in pygame section.*
NPZ Data	*The NPZ data file is a special form in NUMPY, and it is useful to store matrix-like data. In our program the NPZ data file is used to store the social relationship of agents as they move and interact. The computational results of egress flow fields are also saved in NPZ form. The filename are vel_flow1.npz and vel_flow2.npz for Solver 1 and Solver 2, respectively.*

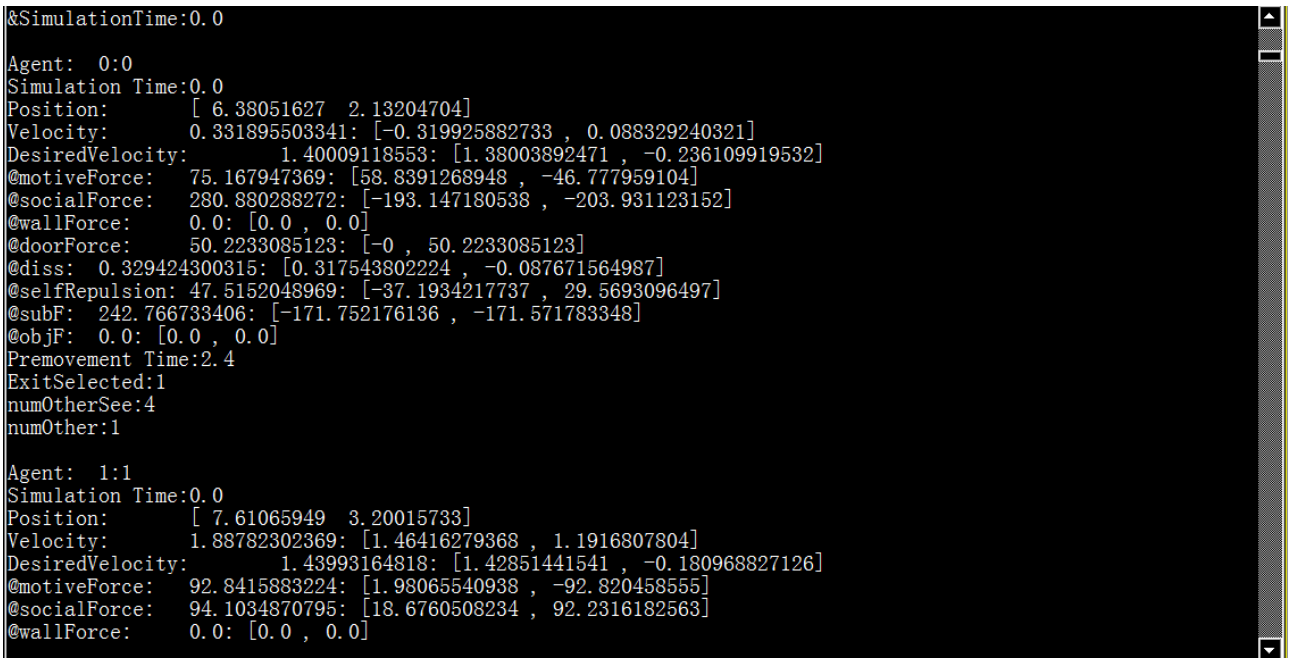


Figure 16. Output Data in Text File.

Try Examples:

There are currently several examples in the repo of https://sourceforge.net/p/crowdegress or https://github.com/godisreal/crowdEgress. For instance there are standard example of a single room with two exits or three exits, which are basically used to test how agents socially interact and select different exits. The example of a room with one exit is also widely used to test crowd behavior at bottlenecks such as the faster-is-slower effect (Helbing, Farkas, Vicsek, 2000).

There are also some more complicated examples. A typical one is learned and extended from MassEgress project (Pan, 2006), and another one is from our IEEE Conference paper (Wang et. al., 2008). Some FDS+Evac test cases are also included. Users can also learn how to write the csv files from the examples.

In order to run such complicated examples it is suggested that users should first check the mesh parameters. Especially, the total number of x point and y points are to be properly given to build a mesh for flow computation. The flow solver may take some computational time to generate the egress flow field, especially if x point and y point are relatively large to generate a refined mesh.

Acknowledgments

The author is thankful to Timo Korhonen, Kerry Marsh and Vivek Kant for helpful comments on earlier work in University of Connecticut. The author appreciates the research program funded by NSF Grant # CMMI-1000495 (NSF Program Name: Building Emergency Evacuation - Innovative Modeling and Optimization). The program source code and numerical test cases are mainly included online at https://sourceforge.net/p/crowdegress/discussion/general/. If you have any comment or inquiry about the testing result, please feel free to contact me at wp2204@gmail.com or start an issue on the repository.

References

D. Helbing, I. Farkas, T. Vicsek, "Simulating Dynamical Features of Escape Panic." Nature, Vol. 407, pp. 487– 490, 2000.

D. Helbing, P. Molnar, "Social force model for pedestrian dynamics," Physical Review E, vol. 51, no. 5, pp. 4282-4286, 1995.

G. P. Forney, "Smokeview, A Tool for Visualizing Fire Dynamics Simulation Data, Volume I: User's Guide", NIST Special Publication 1017-1 6th Edition, National Institute of Standards and Technology, Gaithersburg, MA, June 2022, 188 p.

T. Korhonen, "Technical Reference and User's Guide for Fire Dynamics Simulator with Evacuation," (FDS+Evac, FDS 6.5.3, Evac 2.5.2), VTT Technical Research Center of Finland, 2016.

T. Korhonen, S. Hostikka, S. Heliovaara, H. Ehtamo, "FDS+Evac: Modelling Social Interactions in Fire Evacuation," Proceedings of the 4th International Conference on Pedestrian and Evacuation Dynamics, February 27-29m 2008, Wuppertal, Germany.

T. I. Lakoba, D. J. Kaup, N. M. Finkelstein, "Modifications of the Helbing-Molnar-Farkas-Vicsek Social Force Model for Pedestrian Evolution, Simulation, Vol. 81, Issue 5, pp. 339-352, May 2005.

K. Lewin, Field Theory in Social Science, New York, Harper, 1951.

K. McGrattan, S. Hostikka, R. McDermott, J. Floyd, C. Weinschenk and K. Overholt, "Fire Dynamics Simulator, User's Guide", NIST Special Publication 1019 6th Ed., National Institute of Standards and Technology, Gaithersburg, MA, 2022, 367 p.

F. Ozel, "Time Pressure and Stress as a Factor During Emergency Egress," Safety Science, Vol. 38, pp. 95-107, 2001.

R. Lovreglio, A. Fonzone, L. dell'Olio, A Mixed Logit Model for Predicting Exit Choice during Building Evacuations, Transportation Research Part A, 2016. DOI: 10.1016/j.tra.2016.06.018.

X. Pan, C. S. Han, K. Dauber, and K. H. Law, "Human and Social Behavior in Computational Modeling and Analysis of Egress," Automation in Construction, Vol. 15, pp. 448-461, 2006.

A. F. Peralta , J. Kertesz, G. Iniguez, Opinion dynamics in social networks: From models to data, 2022.

L. A. Quang, N. Jung, E. S. Cho, J. H. Choi and J. W. Lee, Agent-Based Models in Social Physics, Journal-Korean Physical Society, 2018. DOI: 10:3938/jkps.72.1272.

E. Ronchi, R. Lovreglio, M. Kinsey, A Survey on Evacuation Model Awareness, Usage and Users, 2020.

G. Santos and B. E. Aguirre, A Critical Review of Emergency Evacuation Simulation Models, NIST Workshop on Building Occupant Movement during Fire Emergencies, June 9-10, 2004.

M. A. Staal, "Stress, Cognition, and Human Performance: A Literature Review and Conceptual Framework (NASA/TM – 204-212824)," August 2004, Hanover, MD: NASA Scientific and Technical Information Program Office.

P. Wang and X. Wang, "Social Force in Pedestrian Crowd," arXiv:2105.05146v1 [physics.soc-ph], 2021.

P. Wang, "Understanding Social-Force Model in Psychological Principle of Collective Behavior," arXiv:1605.05146v10 [physics.soc-ph], 2021.

P. Wang, P. B. Luh, S. C. Chang and J. Sun, "Modeling and Optimization of Crowd Guidance for Building Emergency Evacuation," Proceedings of the 2008 IEEE International Conference on Automation Science and Engineering (CASE 2008), Washington, D.C., pp. 328 – 334, August 2008.